Amatérské RADIO

NOSITEL VYZNAMENÁNÍ ZA BRANNOU VÝCHOVU I. A II. STUPNĚ

ŘADA PRO KONSTRUKTÉRY

ČASOPIS PRO ELEKTRONIKU A AMATÉRSKÉ VYSÍLÁNÍ

ROČNÍK XXXVIII/1989 ● ● ČÍSLO 5

V TOMTO SEŠITĚ

AMATÉRSKÉ RADIO ŘADA B

Vydává ÚV Svazarmu ve vydavatelství NAŠE VOJSKO, Vladislavova 26, 135 66 Praha 1, tel. 26 06 51–7. Šéfredaktor ing. Jan Klabal, Redakční radu řídí ing. J. T. Hyan. Redaktor L. Kalousek, OK1FAC. Redakce Jungmannova 24, 113 66 Praha 1, tel. 26 06 51–7, šéfredaktor linka 354, redaktor linka 353, sekretářka linka 355. Ročně vyjde 6 čísel. Cena výtisku 5 Kčs, pololetní předplatné 15 Kčs, Rozšiřuje PNS, v jednotkách ozbrojených sil vydavatelství NAŠE VOJSKO, administrace Vladislavova 26, Praha 1. Rozšiřuje PNS. Informace o předplatném podá a objednávky přijímá každá administrace PNS, pošta, doručovatel a předplatitelská střediska. Objednávky do zahraničí vyřizuje PNS – ústřední expedice a dovoz tisku Praha, administrace vývozu tisku, Kovpakova 26, 160 00 Praha 6. Tiskne NAŠE VOJSKO, s. p., závod 08, 160 05 Praha 6, Vlastina ulice č. 889/23.

Za původnost a správnost příspěvku odpovídá autor. Návštěvy v redakci a telefonické dotazy po 14. hodině. Číslo indexu 46 044.

Toto číslo má vyjít podle plánu 9. 10. 1989.

14. zasedání ÚV KSČ:

Plně podporovat vědeckotechnický rozvoj, prosadit přestavbu vědy a především hutnictví, strojírenství a elektrotechniky

Začátkem června zasedal v Praze po čtrnácté ÚV KSČ, který posoudil plnění usnesení ÚV k vědeckotechnickému rozvoji, která schválil XVII. sjezd strany, 7. a další zasedání ÚV KSČ, a úkoly, které v podmínkách realizace přestavby stojí před celou společností. Šlo o komplex problémů a úkolů, na nichž závisí další rozvoj celé naší společnosti a k nimž patří zejména rychlejší postup v intenzifikaci ekonomiky v závislosti na vědě a technice ve spojení s tvůrčím úsilím lidí (zvyšování produktivity práce, snižování náročnosti výroby), zvyšování exportní schopnosti a úrovně uspokojování potřeb lidí při současně se zvyšující ochraně životního prostředí. Pro nás je zajímavé především to, že nejdůležitější a vpravdě klíčový význam v celé této problematice zaujímá hutnickostrojírenský a elektrotechnický komplex. Na zasedání bylo zdůrazněno, že při řešení všech problémů hraje stále větší roli faktor času, neboť zvládnutí (a to komplexní, nikoli polovičaté) vědeckotechnické revoluce je rozhodující v třídním boji s kapitalismem — a obstát v tomto boji je pro socialismus a zachování světového míru otázkou života.

Při kontrole plnění usnesení z minulých let nedošlo, jak konstatoval ve svém úvodním vystoupení na zasedání tajemník ÚV KSČ František Hanus, přes pozitiva, nesporný pokrok a dosažené dílčí výsledky, k potřebnému obratu, nebylo dosaženo požadovaných výsledků při plnění závěrů 10. zasedání ÚV KSČ o úkolech a rozvoji strojírenství, elektrotechnického a hutnického průmyslu i 5. zasedání ÚV k vědeckotechnickému rozvoji.

K příčinám patří kromě jiného, že se nepodařilo vytvořit takovou vazbu mezi plánem rozvoje vědy a techniky a ostatními částmi státního plánu, aby se národohospodářský plán stal nástrojem technické přestavby národního hospodářství, potřebné strukturální změny postupovaly pomalu, neboť pokračovala zastaralá struktura výroby a pokračovaly tendence k extenzívnímu rozvoji, výrobní podniky nebyly stimulovány k většímu a rychlejšímu využívání výsledků výzkumu a prakticky se nezměnilo postavení tvůrčích pracovníků předvýrobních etap a ani jimi dosahované výsledky. V neposlední řadě je na vině i to, že dosavadní mechanismus spolupráce zemí RVHP (přes přijaté programy) vyčerpal do značné míry své možnosti.

Při bližším pohledu na uvedené nedostatky se ukazuje, že více než 90 % potřeb technického rozvoje bylo zajišťováno vlastní vědeckovýzkumnou základnou při výrazné absenci vnějších zdrojů, jako jsou dovozy, licence, know-how, informace, kooperace apod. V této souvislosti se ukázalo jako velmi potřebné pronikavě rozšířit přístup ke světovým poznatkům pro tvůrčí technickou inteligenci, nepovažovat dovoz technické, odborné a vědecké literatury za luxus, ale za nezbytnost. Je třeba rychle změnit i skutečnost, že na jeden výzkumný úkol připadají zatím v průměru 2 až 3 pracovníci výzkumu. Je třeba se zmínit i o nutnosti vysílat technické kádry v daleko větší míře na zahraniční stáže, stipendia, kongresy či sympózia a kontrolovat efektivnost a „návratnost" těchto akcí. Centrální politika bude proto podporovat otevřenost ekonomiky, její větší zapojení do světové vědeckotechnické revoluce i přísun vědeckotechnických znalostí ze všech teritoriálních oblastí. Je naléhavě nutné podstatně rozšiřovat dovozy nejprogresívnější techniky, moderních strojů i technologických procesů a aktivizovat i náš vývoz. Je ovšem třeba zredukovat počet státních výzkumných a jiných úkolů, soustředit se na vybrané směry a výrobky s tím, že se prioritně bude prosazovat snižování energetické náročnosti rozvoje, postupně redukovat výroba hutnictví železa a prosazovat rozvoj nových progresívních materiálů.

Klíčovou otázkou kvalitativního rozvoje je plnoprávné postavení vývojového konstruktéra a technologa, stejně jako peněžní stimulace tvůrčí technické inteligence vůbec. V celé této oblasti pokračují problémy s nedostatečně diferencovaným odměňováním podle výsledků práce a to především u konstruktérů, projektantů a technologů, což bývá spojeno i se slabým morálním oceněním jejich práce. Zde je třeba se zmínit i o tom, že pracovně sociální jistoty byly a dodnes jsou někde a někdy přeceňovány a přerostly až do sociální bezstarostnosti, s čímž souvisí i rovnostářství v odměňování a poskytování různých výhod apod.

Pokud jde o elektrotechnický průmysl, jeho zaostávání za světovým vývojem a potřebami národního hospodářství zná do jisté míry i každý z nás: na trhu chybí dostatečně bohatý a jakostní sortiment spotřební elektroniky, je nedostatek součástek a to především moderních součástek, jak pasívních, tak aktivních, i když se proti minulým letům poněkud zlepšila sortimentní skladba, stále při návštěvě obchodů slyšíme častěji slůvko „nemáme" než naopak. Tady se ukazují hříchy minulosti nejzřetelněji pro každého z nás — tady si ověřujeme mimořádně negativní důsledky nedostatečného technologického rozvoje a nepředvídavosti či nekvalifikovanosti řídící sféry: průměrné stáří výrobních strojů se v této oblasti pohybuje kolem 18 let! Vzpomene si ještě někdo na počítače a ostatní výrobky, které se vyráběly před 18 lety? Je zřejmé, že na tak starých strojích moderní techniku vyrábět nelze — stačí prostý pohled na „vnitřnosti" dejme tomu laserového gramofonu a např. před 18 lety vyráběného cívkového magnetofonu.

Vzhledem k tomu, že se elektronika a strojírenství musí stát základem pro další rozvoj celého národního hospodářství, byla přijata opatření, jejichž dodržování bude kontrolováno a je předpokladem pro další rozvoj společnosti. V oblasti elektrotechniky to je např. urychlený rozvoj elektronické

součástkové základny, který se, díky své velké náročnosti po stránce investic, v současnosti upřesňuje. Pro další rozvoj efektivnosti má velký význam i státní program rozvoje „Nové materiály a technologie jejich výroby a zpracování“. Na rozdíl od tohoto a dalších prioritních úkolů budou na druhé straně utlumovány programy, které nemají odbytovou perspektivu a programy s velkou energetickou a materiálovou náročností a nízkou ekonomickou efektivností. Bude třeba si zvyknout, že v našich omezených podmínkách nelze doma vyrábět celý sortiment jak elektronických, tak elektrotechnických, strojnických a dalších výrobků, není to možné z kapacitních, materiálových a dalších důvodů — je jen škoda, že jsme se k tomuto poznání propracovali tak pozdě.

Východiska a řešení problémů lze najít ve Stanovisku ÚV KSČ, které bylo přijato na závěr 14. zasedání ÚV KSČ. Na základě hodnocení potřeb současného vývoje společnosti a přípravy XVIII. sjezdu KSČ považuje za nezbytné dopracovat a realizovat racionální organizaci a činnost jednotného výkonného národohospodářského centra a vztahy mezi tímto centrem a státními podniky. Rozhodující je urychlování vědeckotechnického rozvoje, který je třeba důsledně spojovat se sociálními, ekonomickými a ekologickými cíli.

V rámci přestavby hospodářského mechanismu podporovat technicky progresívní podniky, uplatňovat výraznou mzdovou diferenciaci v závislosti na skutečně dosahovaných výsledcích. Uplatňovat soubor ekonomických nástrojů tak, aby pro všechny bylo výhodnější a efektivnější životní prostředí zlepšovat, než ho narušovat. Soustředit investice na rychlou obnovu a modernizaci technologických pochodů. Zásadně změnit podmínky práce výzkumných a vývojových ústavů a odborných pracovišť, aby byly závislé na kvalitě a výsledcích práce. Vytvořit podmínky pro rozvoj schopností a uplatnění mladé inteligence. Stanovit a průběžně zpřesňovat strategii státní vědeckotechnické politiky. Urychlit vědeckovýzkumnou činnost atd.

V závěru Stanoviska pak ÚV KSČ požaduje od hospodářských organizací zcela konkrétní opatření: důsledně zajišťovat potřeby vnitřního trhu kvalitními a technicky pokrokovými strojírenskými a elektrotechnickými výrobky pro usnadnění práce v domácnostech a využití volného času již v prvních letech 9. pětiletky. Současně s tím rozšířit a zkvalitnit servisní a opravárenské služby.

Úvod do číslicové a mikropočítačové techniky

František Kyrš, Tomáš Kyrš

Pro vývoj, který v současné době probíhá v moderní elektronice, lze jen stěží hledat paralelu v jiném technickém odvětví — naopak, tato odvětví jsou elektronikou výrazně ovlivňována. K nejdynamičtěji se rozvíjejícím oblastem elektroniky beze sporu patří výpočetní elektronika.

S výpočetní a mikroprocesorovou technikou se dnes dostávají do bezprostředního kontaktu lidé, kteří by jinak neměli s elektronikou (a mnohdy ani s technikou) mnoho společného — konstruktéři, doktoři, vědci, dělníci, učitelky, úřednice... S tím je spojen určitý paradox — tito lidé vědí často o počítačích víc, než mnohý elektronik, který má přece k pochopení činnosti počítače nejblíže. Je však skutečností, že při současném rozsahu elektroniky má každý, kdo není specialista na výpočetní a mikroprocesorovou techniku (ať již profesionál nebo amatér), určité problémy s udržováním dostatečného kontaktu se současným stavem v této oblasti.

Soudobá mikropočítačová technika, to jsou dnes především 16 a 32bitové osobní mikropočítače, kompatibilní nebo blízké standardu PC. Tato skupina je v našich podmínkách výlučně zaměřena uživatelsky. Díky téměř nepřebernému množství dostupných programových „balíků“ mohou být tyto mikropočítače efektivně využívány při nejrůznějších pracích, především kancelářského, konstrukčního, technického i výpočetního charakteru. Jakmile je jednou určitý program zvládnut, může být pro jeho využívání vyškolena většinou i osoba jinak věci neznalá, která pak může podávat i profesionální výkony.

O běžných aplikacích obdobné techniky v průmyslu lze však hovořit spíše u druhé skupiny počítačů, kterou představují různé varianty 8bitových mikropočítačů, především pro příznivější ekonomické ukazatele a také přes všechny problémy zlepšující se dostupnost moderních součástí. Zde máme na mysli zejména očekávané jednočipové mikropočítače řady 8051 a statické paměti CMOS s většími kapacitami. To vše signalizuje blížící se změnu podmínek a tím také odpovídající potřebu změny přístupu k řešení elektronických (ale často i mechanických apod.) obvodů a zařízení.

Dnes je již jisté, že žádný, ani úzce specializovaný amatér nebo profesionál, který nechce zaostat ve svém vývoji, nemůže číslicovou a mikropočítačovou techniku nechat na okraji zájmu. Musí si vytvořit alespoň základní orientaci v jejích principech, možnostech a směrech předpokládaného vývoje. Přemýšlet o uplatnění možností, které se již ukazují být reálné, spolupracovat i se specialisty z jiných oborů. Právě taková „druhá vlna“, skuteční aplikátoři mikroprocesorové techniky, s novými, dobrými nápady na její uplatnění, by měla přinést ten pravý technický i ekonomický pokrok. Nadšenci, kteří stáli a stojí v první vlně rozvoje této techniky, vyčerpávali a vyčerpávají příliš mnoho své energie na základní problémy. Mají však velké zkušenosti, které lze od nich čerpat.

Každý, kdo se nyní rozhoduje začít se číslicové a mikroprocesorové technice věnovat soustavně, si musí uvědomit, že je to dlouhodobá, trvalá záležitost. Musí si vytvořit svůj vlastní styl práce, číst co se dá, shánět informace, třídit je a získávat přehled o reálné situaci. K základům také patří alespoň orientační zvládnutí některého vyššího programovacího jazyka (snad Pascalu, který je dobrým východiskem pro práci s asemblerem) a také angličtiny. Není třeba žádné dokonalosti, lámaná technická angličtina stačí a tu lze zvládnout za několik měsíců. Posledním předpokladem je pochopitelně možnost praktické práce, nejlépe v partě šikovných lidí. Pak to chce již jen hledat nápady a přemýšlet, jak je realizovat. Nejlepší předpoklady k úspěchu mají lidé s tvůrčím elánem, nezatížení balastem „zkušeností“, z nichž mnohé již po několika letech bývají vyčichlé a k ničemu.

Úvod do celé problematiky jsme se pokusili usnadnit dvěma obsahově navazujícími čísly AR řady B, z nichž první právě otevíráte, druhé vyjde v příštím roce. Koncepce obou příspěvků byla volena tak, aby jejich studium vyžadovalo minimální předběžnou přípravu — předpokládá se úroveň běžného, univerzálně orientovaného čtenáře AR, jehož znalosti vyplývají především z praktických zkušeností se stavbou jednoduchých „číslicových“ konstrukcí.

V tomto čísle jsme se snažili nejprve vytvořit pokud možno systematický pohled na kombinační a sekvenční logiku. Dále, po stručném přehledu číslicových a paměťových obvodů, se věnujeme základům hardwarových a programovatelných automatů. Tím si vytváříme předpoklady pro výklad principů, techniky a programování mikroprocesorových systémů a mikropočítačů, který bude obsahem dalšího čísla.

I. Dvoustavová (binární) logika

Skutečným základem prakticky všech digitálních zařízení, řídicích jednotek, programovatelných automatů, kalkulátorů, počítačů atd. je tzv. binární logika. Přitom nelze říci, že by byla něčím novým, vždyť například Booleovo stěžejní dílo „Investigation of the laws of the thought — Zkoumání zákonů myšlení“ bylo vydáno již v polovině

minulého století. Existují ovšem i jiné, například třístavové logické systémy.

Technický úspěch binární logiky má jednoduché důvody. Prvním je její univerzálnost a dokonalé teoretické zvládnutí. Druhým a podstatným, který navzdory technickému pokroku vlastně od počátků automatizace a kybernetiky nebyl oslaben, je možnost efektivně realizovat fyzikálně logické součástky — dokonce stále více platí, že spolehlivé, hromadné a levné výroby s velkou výtěžností a hustotou integrace (postupně kontakt, relé, elektronka, tranzistor, integrovaný obvod . . .) lze dosáhnout u jednotlivých součástek pouze při orientaci na využívání jejich dvou mezních stavů, tj. aktivní/pasívní, zapnuto/vypnuto . . .

Teorie binární logiky se velmi často podceňuje, svádí k tomu právě zdánlivá jednoduchost rozlišení možných dvou stavů a také velmi často lehce nabyté první zkušenosti z praktické práce. Ve skutečnosti je to však složitější — znát základní funkci hradla, děliče a monostabilního obvodu už dnes nestačí. Současná technika těží především z možností velkého stupně integrace (obvody LSI). Funkce většiny obvodů není příliš průhledná, často se může měnit mezi několika módy či režimy podle různých podmínek, programování apod. Soubory jednotlivých obvodů pak vytvářejí různé funkční bloky, mezi nimiž se uskutečňují vzájemné komunikace a vazby na základě různých principů a konvencí. K získání nezbytné suverenity je i při amatérské práci velmi užitečné zvládnout jak systémový, „zastřešující" pohled na oblast číslicové techniky, tak i teoretické základy binární logiky. Té jsme se rozhodli věnovat první část tohoto příspěvku, shrnující základní principy, jejich souvislosti a jednoduché metody řešení logických sítí, které by mohly tvořit protiváhu k většinou intuitivním postupům.

Jako každá logika, má i logika binární mnoho společného s lidským myšlením. Setkáváme se v ní s rozhodováním, vylučováním, vyplýváním, negací a jinými operacemi. Tyto operace jsou pro každý druh logiky (viz např. množinová logika) definovány specifickým způsobem. Matematickou definicí, popisem a řešením binární logiky se zabývá Booleova algebra.

Booleova algebra

V úvodu hned zdůrazněme, že Booleeova algebra není algebrou čísel, s jakou se setkáváme v matematice. Je to algebra stavů. Vzhledem ke klasické algebře je proto jinak definována, např. v ní vůbec nenacházíme operace odčítání a dělení (ty ve stavové algebře neexistují).

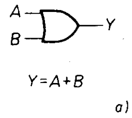
Základní funkce Booleovy algebry jsou:

a) logický součet (disjunkce),
b) logický součin (konjunkce),
c) negace.

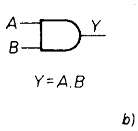
Tyto tři funkce si nyní podrobně popíšeme. Nejprve ještě učiňme dohodu, že v celé kapitole budeme proměnné označovat velkými písmeny bez indexů. Rovněž oba logické stavy, logickou jednotku a nulu budeme zapisovat symboly 1, 0. Pro operátory logického součtu a součinu budeme užívat běžné symboly (+) a (.).

a) Logický součet (OR)

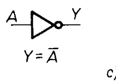
A + B = Y (1),
výstupní proměnná Y má hodnotu



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



A	Y
0	1
1	0

Obr. 1. Základní logické funkce a logické členy; a) symbol a pravdivostní tabulka logického součtového hradla OR, b) symbol a pravdivostní tabulka logického součinového hrdla AND, c) symbol a pravdivostní tabulka logického invertoru

1 tehdy, má-li alespoň jedna ze vstupních proměnných A, B hodnotu 1. Symbol a pravdivostní tabulka OR je na obr. 1a.

b) Logický součin (AND)

A . B = Y (2),
výstupní proměnná Y má hodnotu 1 tehdy, mají-li hodnotu 1 obě vstupní proměnné A, B, obr. 1b.

c) Negace

$\overline{A}$ = Y (3),
výstupní proměnná Y má vždy vůči proměnné vstupní A opačnou hodnotu, je její inverzní funkcí (negací), obr. 1c.

Na obr. 1 jsme poprvé použili pravdivostní tabulku. Její standardní formát umožňuje vzájemně přiřadit vstupní a výstupní proměnné jak u základních, tak u složitých kombinačních funkcí. To oceníme při návrzích obvodů, které pro nás v době sestavování tabulky často představují pouze „black box", černou skříňku.

Uvedené tři základní funkce lze rozšířit na libovolný počet vstupních proměnných a to v přímém i inverzním tvaru. Kombinací takových funkcí pak vznikají obecné logické rovnice pro *n* proměnných. Uveďme si nyní některá pravidla pro práci s Booleovou algebrou. Jsou většinou formálně shodná s pravidly vžité, číselné algebry. Přesto pozorujeme (nebo si spíše většinou neuvědomíme) některé odchylky, vyplývající z omezení hodnoty proměnné na dvě nespojité, diskrétní úrovně 1, 0.

Zcela logická jsou pravidla pro součet a součin jedné proměnné s konstantou. Tyto i ostatní vztahy si lze ověřit dosazením hodnot 1 a 0 za proměnnou. Již při dvou proměnných však vidíme, jak je takový důkaz (postupné vyčerpání všech možných kombinací logických hodnot proměnných) zdlouhavý:

A + 0 = A   A.0 = 0
A + 1 = 1   A.1 = A
A + A = A   A.A = A   (4).
A + $\overline{A}$ = 1   A.$\overline{A}$ = 0

Pravidla pro operace s několika proměnnými se řídí podobnými zákony, jaké známe. Zkuste si je projít, jsou uvedené vždy společně pro logický součet a součin:

komutativní zákony
(1.) A + B = B + A
(2.) A.B = B.A (5),

asociativní zákony
(1.) (A + B) + C = A + (B + C)
(2.) (A.B).C = A.(B.C) (6),

distributivní zákony
(1.) (A + B) . C = A.C + B . C
(2.) (A + C) . (B + C) = AB + C (7).

Všechny zákony jsou jistě naprosto jasné, od definice číselné algebry se však jeden z nich zcela odlišuje. Je to druhý z distributivních zákonů (pro násobení). Vysvětlíme si to v následujícím příkladu:
(A + C) . (B + C) = AB + BC + AC + C;
podle prvního distributivního zákona pro sčítání upravíme rovnici do tvaru
(A + C) . (B + C) = AB + (A + B) C + C,
v němž výraz (A + B) C může při A + B = 0, popř. 1 nabývat hodnoty (A + B) . C = 0, popř. C. Proto platí, že AB + 0 (popř. C) + C = AB + C a lze psát rovnost
(A + C) . (B + C) = AB + C,
shodnou s definicí 2. distributivního zákona.

Dokazovat obdobným způsobem správnost tohoto zákona při konverzi v opačném směru by bylo obtížnější. Důkaz přineseme později s využitím Shannonova teorému;

zákon dvojnásobné negace
$\overline{\overline{A}}$ = A (8),
po dvojnásobné negaci je výstupní proměnná totožná se vstupní proměnnou.

Zcela mimořádnou vlastností Booleovy algebry je její dualita. Vyplývá z naprosté symetrie základních zákonů — ke každému zákonu lze vždy najít zákon další, k původnímu duální. Duální zákon lze přitom odvodit z původního poměrně jednoduchým postupem, založeným na využití principu inverzní funkce. Důsledkem duality je to, že libovolnou logickou funkci můžeme volbou vhodného postupu vyjádřit v jiném, duálním tvaru. Těmito problémy se budeme zabývat dále. Dříve si však řekněme, které zákony jsou vzájemně duální: disjunkční a konjunkční funkce jedné proměnné, stejně jako první a druhé komutativní, asociativní a distributivní zákony. Vzájemně duální zákony lze použít k tomu, abychom libovolný logický výraz vyjádřili v jeho duálním tvaru. Přitom je nutné postupovat podle určitých pravidel, v nichž klíčovou roli hraje již zmíněný princip inverze logické funkce. Zvládnutí těchto pravidel, formulovaných de Morganovými zákony a v zobecněné formě Shannonovým teorémem, je velmi užitečné, protože na nich založený postup odvození duálního logického výrazu nevyžaduje důkazu. Proti intuitivnímu přístupu ke konkrétním úlohám, které většinou zdaleka nemají jediné řešení, se tak práce velmi zjednodušuje a omezuje se možnost zavádění chyb do výpočtu, zvláště při větším počtu proměnných.

De Morganovy zákony

V praxi nejrozšířenější využití principu duality představují de Morganovy zákony, umožňující mechanicky určit duální funkce k negovanému součtu nebo součinu libovolného počtu proměnných, definovaných v přímém tvaru. Naproti tomu je duální výraz tvořen přímým součinem nebo součtem proměnných, které jsou vůči výrazu originálnímu definovány ve tvaru inverzním.

De Morganovy zákony jsou tedy dva. Platí

$\overline{A.B.C\ldots} \rightleftarrows \overline{A}+\overline{B}+\overline{C}\ldots$

$\overline{A+B+C\ldots} \rightleftarrows \overline{A}.\overline{B}.\overline{C}\ldots$ (9)

Vidíme, že de Morganovy zákony definují přímé duální funkce, levé i pravé strany rovnic jsou duální, každá popisuje zcela shodnou funkci, ale jiným způsobem.

Z definice zákonů je patrno určité omezení. Na obou stranách rovností se vždy vyskytují všechny proměnné v jediném tvaru, buď přímém, nebo negovaném a na každé straně je vždy jediný typ operátoru. Jak uvidíme dále, princip de Morganových zákonů lze snadno zobecnit pro libovolné logické výrazy.

Formát de Morganových zákonů je mimořádně vhodný k odvození a zapamatování dvou důležitých skutečností. Pro jejich zvýraznění jsme ve vztazích (9) nahradili znaménka rovnosti šipkami.

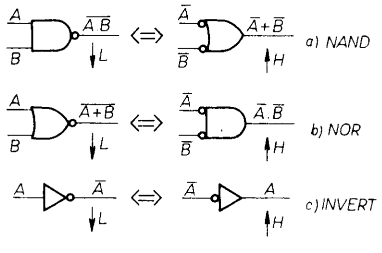
Z postupu stanovení duálního výrazu zleva doprava je podle obou zákonů patrné, že:

a) každé hradlo NAND lze nahradit hradlem OR za předpokladu, že jeho vstupní signály budou negovány příslušnými invertory na vstupech,

b) každé hradlo NOR lze nahradit hradlem AND, opět doplněným příslušnými vstupními invertory.

Příklady realizace obou duálních logických obvodů jsou na obr. 2. Pro doplnění je uveden i princip náhrady hradel OR, AND hradly NOR, NAND pomocí negace výstupní funkce doplňkovým, výstupním invertorem.

Druhou praktickou zkušenost lze získat přiřazením schematických symbolů duálních funkcí z pravé strany vztahu (9) ke klasickým symbolům funkcí NAND, NOR. Postup je jednoduchý. Symbol hradla NAND/NOR svým kroužkem (non) na výstupu udává, že jde o ekvivalent hradla AND/OR s invertorem na výstupu. Obdobně lze proto sestavit symboly duálních obvodů s hradlem OR/AND (podle de Morganových zákonů i obr. 2), nyní ovšem se znaky inverze na vstupech příslušného hradla. Tedy, libovolnou základní logickou funkci lze definovat vždy dvěma symboly. Jeden z nich využívá součinového, druhý součtového hradla. Ačkoli je jejich funkce zcela stejná, lze podle toho, zda je výstup symbolu přímý nebo invertovaný, naznačit (při kreslení schématu) nebo poznat (při jeho čtení), jakou úlohu má konkrétní hradlo v zapojení, zda je na jeho výstupu rozhodující dosažení hodnoty 0 (symbol NON) nebo 1 (přímý výstup). Pak je již, podle typu hradla (OR, AND) a přímého nebo invertovaného charakteru vstupů, snadné odvodit potřebnou podmínku ze strany vstupních signálů. Příklady duálních symbolů pro funkce z de Morganových zákonů jsou na obr. 3. Stejně užitečná je i úprava symbolu



Obr. 3. Duální symboly shodných funkcí s vyznačením skutečně aktivní výstupní úrovně hradla

invertoru. Ačkoli podobné kreslení obvodů naše norma nepřipouští, je užitečná jejich znalost při studiu zahraničních pramenů (např. Intel..) nebo pro vlastní potřebu.

Pokračujme však dále. I když de Morganovy zákony názorně objasňují princip duality Booleovy algebry, nejsou ve své základní formě dostatečně univerzální, neumožňují přehlednou práci se složitějšími výrazy nebo proměnnými v obecném tvaru.

Shannonův teorém

Zobecněné využití principů duality, které si např. při práci s de Morganovými zákony musíme odvozovat často velmi těžkopádně, definuje velmi jednoduše Shannonův teorém. Tento mimořádně praktický zákon umožňuje zcela mechanicky stanovit inverzní funkci libovolného výrazu podle vztahu

$f[A,\overline{B},C,\ldots,(+),(.)] =$
$= \overline{f}[\overline{A},B,\overline{C},\ldots,(.),(+)]$ (10)

— inverzní logická funkce je z funkce původní odvozena tak, že:

a) invertujeme každou původní proměnnou,

b) vzájemně zaměňujeme všechny logické operátory, tj. (+) nahradíme (.) a opačně.

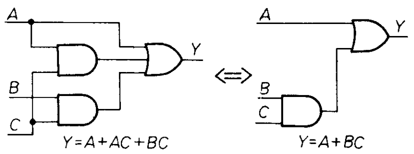
Zde je třeba jasně definovat, jaký je rozdíl mezi duální a inverzní funkcí. Zatímco duální funkce je jiným způsobem vyjádřená, avšak obsahově zcela shodná s funkcí původní, je inverzní funkce skutečnou inverzí funkce původní, je její negací. Jak vyplývá z definice de Morganových zákonů i Shannonova teorému, při vytváření duální i inverzní funkce se vzájemně mění disjunkční výrazy na konjunkční a opačně. Z toho vyplývá, že duální formu libovolného logického výrazu můžeme odvodit posloupností stanovení inverzní funkce a její následnou negací. Přitom lze funkce výhodně upravovat a zjednodušovat při dodržování pravidel Booleovy algebry. Navíc přistupuje nutnost věnovat zvýšenou pozornost prioritě logických operátorů — násobení má přednost před sčítáním. Nejbezpečnější je před začátkem jakýchkoliv úprav zavést do výrazu explicitní závorky. Při inverzích složitých výrazů s několikastupňovými negacemi je nutné jejich postupné zjednodušení směrem zevnitř.

Postup při použití Shannonova teorému nejlépe osvětlí jednoduché příklady.

1. Nejprve znovu ověříme 2. distributivní zákon:
(A + C) . (B + C) = Y.
Inverzí funkce podle Shannona získáme součtový tvar
$\overline{A}\overline{C} + \overline{B}\overline{C} = \overline{Y}$,
který využitím 1. distributivního zákona upravíme
$\overline{C}(\overline{A} + \overline{B}) = \overline{Y}$
a nakonec zpětnou inverzí
C + AB = Y
získáme duální funkci a tedy důraz správnosti.

2. Také důkaz v opačném směru bude zcela jednoduchý:
(AB) + C = Y.
Inverzní funkce
$(\overline{A} + \overline{B}) . \overline{C} = \overline{Y}$,
po úpravě podle 1. distributivního zákona
$\overline{A}\overline{C} + \overline{B}\overline{C} = \overline{Y}$
a opětovné inverzi získáme opět správný tvar duální funkce
(A + C) . (B + C) = Y.

3. Jako další příklad zkusme minimalizovat, tj. co nejvíce zjednodušit funkci
A + AC + BC = Y.
Nejprve z opatrnosti zavedeme prioritní závorky
A + (AC) + (BC) = Y,
pak vytknutím proměnné C upravíme
A + C (A + B) = Y
a určíme inverzní funkci
$\overline{A}(\overline{C} + \overline{A}\overline{B}) = \overline{Y}$.
Protože $\overline{A}.\overline{A} = \overline{A}$, lze upravit
$\overline{A}\overline{C} + \overline{A}\overline{B} = \overline{Y}$
a po vytknutí
$\overline{A}(\overline{B} + \overline{C}) = \overline{Y}$
shledáváme, že výraz už dále zjednodušit nelze. Proto opětovnou inverzí zjistíme konečný tvar minimalizované funkce
A + BC = Y,
porovnání realizace původní a minimalizované funkce s hradly AND, OR je na obr. 4.

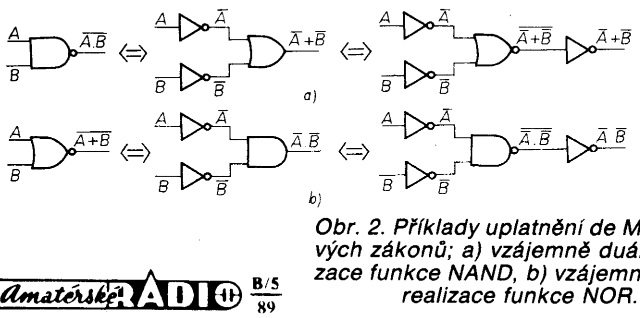


Obr. 4. Schematické znázornění realizace přímé a minimalizované funkce

4. Jako poslední příklad minimalizujme různými postupy funkci $Y = A + \overline{AB}$, tj. disjunkční funkci členů v přímém a inverzním tvaru:

a) bez předběžné úpravy negovaného součinu:
$Y = A + (\overline{AB})$,
$\overline{Y} = \overline{A} . AB$,
protože $\overline{A}.A = 0$,
$\overline{Y} = 0.B = 0$,
je po opětovné inverzi
Y = I;

b) s využitím úpravy $\overline{AB}$ podle de Morganova zákona:
$Y = A + \overline{AB}$.
$Y = A + \overline{A} + \overline{B}$,
protože $A + \overline{A} = I$, bude
$Y = I + \overline{B}$,
Y = I.



Obr. 2. Příklady uplatnění de Morganových zákonů; a) vzájemně duální realizace funkce NAND, b) vzájemně duální realizace funkce NOR.

Vidíme, že v tomto případě nebylo třeba inverzní funkci vůbec hledat.

Závěrem této části konstatujeme, že Shannonův teorém umožňuje snadno odvodit inverzní tvar libovolné logické funkce. Ta je, díky vzájemné konverzi disjunkčních a konjunkčních tvarů, velmi užitečná pro přehledné úpravy a minimalizace logických výrazů i funkcí.

Se znalostí dosud uvedených zákonů již lze vystačit při řešení mnoha praktických problémů v různých oblastech aplikace kombinačních obvodů. Bohužel, složitost jejich řešení, nepřehlednost a obtížnost algebraického popisu funkce exponenciálně narůstá s počtem proměnných v logické rovnici. Jen pro orientaci, možný počet funkcí n proměnných je úměrný vztahu $F_n=(2)^{2^n}$, tj. pro tři proměnné se již rovná 256. Proto se často dostáváme do bludného kruhu, k rozsáhlým rovnicím s velkým nebezpečím zavedení početních chyb a těžkému hledání optimálního řešení. To proto, že většina úloh má několik alternativních řešení.

Než pokročíme dále, všimněme si ještě několika pravidel Booleovy algebry, nazývaných pravidly pohlcování, která často mohou logický výraz podstatně zjednodušit. Jak snadno je lze dokázat, tak snadno lze jejich uplatnění v rovnici přehlédnout. Platí:

$A + \bar{A}B = A$,

výsledek funkce je plně určen proměnnou A, na konjunkci AB vůbec nezávisí;

$A\,(A + B) = A$,

zde je funkce určena konjunkcí $A.A=A$;

$(A + \bar{B})\,B = AB$,

protože $B.\bar{B}=0$, je funkce plně určena konjunkcí AB;

$A\bar{B} + B = A + B$,

je-li proměnná $A=0$, je funkce určena proměnnou B,

je-li naopak $B=0$, je funkce určena proměnnou A.

Shannonův rozvoj

Většina návrhů a řešení logických obvodů v praxi začíná sestavením pravdivostní tabulky. Teprve tabulka, která je popisem požadované reakce logické sítě na různé vstupní podmínky (kombinace vstupních proměnných), je podkladem pro sestavení logických rovnic. Z již naznačených důvodů se snažíme algebraickým řešením vyhýbat nebo je omezit na únosnou míru. Jednou z velmi vhodných cest představuje užívání tzv. Karnaughovy mapy, která na základě určitých pravidel představuje grafickou interpretaci pravdivostní tabulky. Mapa umožňuje podstatnou část algebraického řešení logické rovnice nahradit efektivnějším a mnohem přehlednějším řešením „topografickým". Pochopit princip Karnaughovy mapy, jejího formátu, zápisu proměnných i vlastní práce s ní pomůže princip Shannonova rozvoje.

Jeho smyslem je především možnost rozložit složité logické funkce podle zvolené proměnné do soustavy dvou funkcí. Přitom se proměnná z obou funkcí vytýká a tak v nich není obsažena, tím se zmenší počet proměnných o jednu, jde tedy o podstatné zjednodušení. Funkci lze přitom rozkládat jak podle libovolné proměnné, tak postupně podle všech proměnných. Z praktických důvodů se operace používá téměř výlučně u funkcí v disjunkčním tvaru. Obecný postup rozkladu podle jedné proměnné (např. A) naznačuje vztah

$$f_A[A,B,C\ldots] = A.f[1,B,C\ldots] + \bar{A}.f[0,B,C\ldots] \quad (11),$$

logickou funkci podle určité, zvolené proměnné rozložíme tedy tak, že ji v tom tvaru, v jakém se vyskytuje v původním výrazu, vytkneme před první výraz a v negovaném tvaru před výraz druhý. Do obou výrazů opíšeme původní tvar s tím rozdílem, že do prvního zapisujeme konstantu I, do druhého 0 za všechny výskyty proměnné (ať v přímém nebo inverzním tvaru), podle které se provádí rozvoj. Obsah funkce se rozvojem nemění.

Po úplném rozvoji (11) podle jedné proměnné získáme tzv. *disjunkční normální tvar* funkce nebo výrazu — DNT. Postup je v tomto případě přesně opačný než při zjednodušování výrazu. Nevyužíváme například operací jedné proměnné ap., abychom ve funkci DNT získali co největší počet konjunkcí.

Jestliže budeme v rozkladu systematicky pokračovat až do vyčerpání všech proměnných (A,B,C...), rozložíme nakonec funkci do tzv. *úplného normálního disjunkčního tvaru* — UDNT. Ani nyní se obsah logické funkce nemění.

Pro úplnost je třeba dodat, že obdobná pravidla platí pro duální, konjunkční rozvoj. V praxi se s výhodou užívá možnosti vzájemné konverze DNT/KNT a UDNT/UKNT využitím inverze funkce Shannonovým teorémem a její následnou negací.

Disjunkční rozvoj si ukážeme na příkladu jednoduchého logického součtu dvou proměnných $A + \bar{B}$.
Nejprve výraz rozložíme podle proměnné A,

$$f_{DNT(A)}\,[A + \bar{B}] = A(1 + \bar{B}) + (0 + \bar{B}) = A + A\bar{B} + \bar{A}\bar{B} \quad (12a)$$

stejným postupem rozložíme výraz i podle B,

$$F_{DNT(B)}\,[A + \bar{B}] = \bar{B}(A + 1) + B\,(A + 0) = A\bar{B} + \bar{B} + AB \quad (12b).$$

Lze odvodit, např. minimalizací, že oba hořejší rozvoje (obě funkce DNT) jsou obsahově shodné a odpovídají původní funkci. Vidíme, že DNT tvoří určitý počet vzájemně různých konjunkcí, které mají obecně různý počet členů (menší nebo rovný počtu vstupních proměnných) a že podle zvolené proměnné a nakonec i podle početního postupu mohou mít rozvoje různá řešení.

Úplný disjunkční normální tvar UDNT získáme navazujícím rozvojem DNT podle zbývající proměnné. Vyjdeme-li např. z (12a), pak zbývá udělat rozvoj podle proměnné B. Proto

$$f_{UDNT(A,\bar{B})}\,[A + A\bar{B} + \bar{A}\bar{B}] = \bar{B}\,(A + A.1 + \bar{A}.1) + B\,(A + A.0 + \bar{A}.0) = A\bar{B} + \bar{A}\bar{B} + AB \quad (13a);$$

pokračujeme-li naopak v rozvoji z funkce (12b), zbývá provést rozvoj podle proměnné A

$$f_{UDNT(B,A)}\,[A\bar{B} + \bar{B} + AB] = A\,(1.\bar{B} + \bar{B} + 1.B) + \bar{A}\,(0.\bar{B} + \bar{B} + B) = A\bar{B} + AB + \bar{A}\bar{B} \quad (13b).$$

UDNT je na rozdíl od DNT charakteristický tím, že pro každou funkci má právě jedno jediné řešení (porovnej 13a,b) — právě to je důležité pro pochopení systému Karnaughovy mapy. Má opět určitý konečný počet vzájemně různých konjunkcí, vázaných operátory logického součtu. Jednotlivé konjunkce však nyní mají všechny přesně stejný počet členů, který je roven počtu proměnných původní funkce. Každá proměnná se tak podílí na jednoznačném určení konjunkce, jejímž je členem — z hlediska Karnaughovy mapy určuje jednu její „souřadnici". Jednotlivé konjunkce v UDNT se nazývají mintermy. V rovnicích o dvou, třech či čtyřech proměnných mají minterny vždy dva, tři nebo čtyři členy. Tím je také určen počet souřadnic, nutných pro jednoznačné určení polohy mintermu na Karnaughově mapě.

Mnohý čtenář se jistě ptá, proč jsme se vlastně Shannonovým rozvojem zabývali. V uvedených příkladech jsme sice odvodili zajímavou funkci dvou proměnných, ale v podstatně složitějším tvaru, než byla funkce původní. To je naprostá pravda, podobné úpravy funkcí by nám nic užitečného nepřinesly. Získané poznatky nám však nyní umožní pochopit systém Karnaughovy mapy a její souvislost s algebraickým řešením logické funkce. Výhodou bude, že nyní můžeme postupovat obráceně. Budeme hledat minimalizaci (nám již známou) funkce UDNT z (13) a to jednak pomocí Booleovské algebry, jednak pomocí Karnaughovy mapy, kterou si sami definujeme.

Pravdivostní tabulka, logická funkce, Karnaughova mapa

Předpokládejme, že jsme na základě analýzy požadavků na funkci konkrétního logického obvodu sestavili pravdivostní tabulku na obr. 5. Tabulka popisuje logickou funkci dvou proměnných A,B, jsou tedy možné $2^n = 4$ vzájemné kombinace jejich logických

	Vstupní proměnné		*Výst. proměnná*
	A	*B*	*Y*
$\bar{A}\,\bar{B}$	0	0	1
$\bar{A}\,B$	0	1	0
$A\,\bar{B}$	1	0	1
$A\,B$	1	1	1

Obr. 5. Příklad pravdivostní tabulky

hodnot. Proto má tabulka celkem čtyři řádky. Protože uvažujeme disjunkční funkci, budou mít jednotlivé kombinace konjunkční tvar, $\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$, AB. Jsou zapsány v prvním sloupci tabulky. Pro obě vstupní proměnné jsou vyhrazeny dva sloupce tabulky. Zapisují se v přímém tvaru, to znamená, že platnou inverzní proměnnou označujeme hodnotou 0. Do jednotlivých řádků posledního sloupce zapisujeme dílčí hodnoty výstupní funkce, odpovídající našim požadavkům na funkci logického obvodu.

Pokud některá funkce (některý řádek tabulky) nemusí být vyhodnocena, je vhodné odlišit příslušnou výstupní proměnnou vhodným symbolem, např. X. V takovém případě se jedná o neúplně definovanou tabulku, která může podstatným způsobem zjednodušit jak algebraickou, tak Karnaughovu minimalizaci. Všimněme si ještě, že všechny konjunkce vstupních proměnných ve všech řádcích tabulky mají shodný počet členů, rovný počtu proměnných — jsou to mintermy!

Algebraický zápis disjunkční funkce podle tabulky na obr. 5 je mechanickou záležitostí. Funkce je určena logickým součtem dílčích konjunkcí vstupních proměnných ze všech řádků, pro které má výstupní proměnná nabývat hodnoty I. Proto

$Y = \bar{A}\bar{B} + A\bar{B} + AB$.

Při minimalizaci rovnici nejprve upravíme

$Y = A(\bar{B} + B) + \bar{A}\bar{B}$,

pak využitím zákona $\bar{B} + B = 1$ zjednodušíme $Y = A + \bar{A}\bar{B}$

a úpravou na inverzní funkci zpřehledníme další možný postup

$\bar{Y} = \bar{A}(A + B)$.

Nyní vidíme možnost uplatnění pravidla $\bar{A}.A = 0$, proto

$\bar{Y} = \bar{A}B$.

Nakonec, opakovanou Shannonovou inverzí dostaneme výsledek

$Y = A + \bar{B}$,

shodný s funkcí, na které jsme si ukazovali příklad Shannonova rozvoje. Vidíme, že i když jde o jednoduchou úlohu, lze ji řešit několika způsoby. Ve složitějších případech je často těžko určit, která vede k lepšímu výsledku. Pro více než dvě proměnné je téměř vždy výhodnější použít mapu.

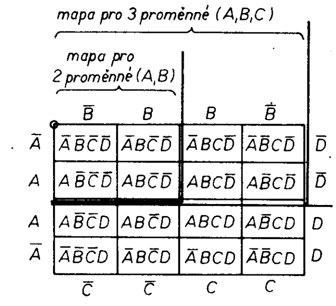
Pochopit princip mapy již bude snadné. Každá mapa se skládá z určitého počtu ohraničených políček. Každé políčko mapy představuje jeden řádek pravdivostní tabulky, tj. současně i jeden z množiny možných mintermů funkce UDNT. Do mapy však může být zapsána i obecná logická funkce. Jedinou podmínkou je to, že musí být upravena do tvaru DNT. Není tedy třeba ji rozkládat na úroveň mintermů. Stačí, když jsou z ní odstraněny všechny závorky. Tvar mapy pochopitelně v každém případě musí odpovídat plnému počtu proměnných logické funkce nebo pravdivostní tabulky.

Karnaughova mapa umožňuje:

— zápis disjunkční funkce nebo pravdivostní tabulky,
— její minimalizaci nebo jiné logické úpravy, příkladem možných úprav je rozvoj funkce až do úrovně UDNT,
— inverzi funkce,
— určení duální funkce, vzhledem k zápisu zpravidla v konjunkčním tvaru.

Vidíme, že díky mapě můžeme realizovat prakticky všechny operace, jimiž jsme se dosud zabývali na základě Booleovy algebry. Z toho ovšem vyplývá, že mezi algebraickým vyjádřením funkce, pravdivostní tabulkou a Karnaughovou mapou musí existovat jednoznačný systém vzájemného přiřazení jednotlivých proměnných a logických operátorů. Je zajištěn na úrovni elementárních konjunkcí UDNT, které jsou v mapě topologicky seřazeny v definovaném pořadí podle principu tzv. sousedních mintermů — jednotlivá políčka mapy (sousední mintermy) se ve všech směrech (nahoru i dolů, doleva i doprava) mohou a musí odlišovat jedno od druhého pouze inverzí jedné jediné proměnné. Tím je zcela zaručen definovaný souřadný systém mapy.

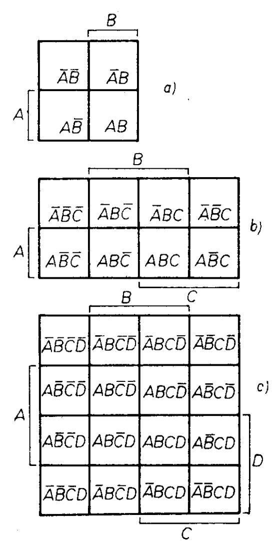
V praxi se užívají mapy pro $n = 2$, 3 a 4 proměnné s odpovídajícím počtem mintermů UDNT (políček mapy) $2^n = 4$, 8 a 16. Aby byla zajištěna



Obr. 6. Symbolické znázornění konstrukce Karnaughovy mapy pro čtyři proměnné (A, B, C, D) a jejích podmnožin pro tři (A, B, C,) a dvě (A, B) proměnné

vzájemná kompatibilita, orientace ve všech mapách různého původu a nakonec také přehlednost vlastní práce, dodržuje se zásada umisťování mintermu se všemi proměnnými v inverzním tvaru $\bar{A}$ $\bar{B}$ ($\bar{C}$) ($\bar{D}$) do levého horního rohu mapy. Tím je definována jednotná struktura mapy pro libovolný počet proměnných (obr. 6). Pro názornost jsou v každém políčku vyznačeny tvary (přímý, inverzní) všech proměnných příslušného mintermu. Pro mapu o dvou proměnných samozřejmě platí pouze proměnné A, B, pro mapu o třech proměnných proměnné A, B, C.

Ve skutečnosti ovšem prostor uvnitř každého políčka slouží k vepsání logické hodnoty jeho mintermu, adresa políčka je určena souřadným systémem proměnných, vyznačených vně mapy. Přitom se zapisují pouze proměnné v přímém tvaru, za jejich inverze se automaticky považují všechna zbylá, neoznačená políčka na stejné vnější straně mapy. Dvěma, třemi nebo čtyřmi souřadnými symboly (= proměnné v přímém tvaru) je tak dokonale definována pozice každého mintermu v mapě pro dvě, tři nebo čtyři proměnné. Celý systém je pro všechny případy samostatně znázorněn na obr. 7.



Obr. 7. Formáty a označení Karnaughovy mapy pro 2, 3 a 4 proměnné; a) mapa dvou proměnných (odpovídá dvourozměrnému, plošnému znázornění), b) mapa tří proměnných (odpovídá třírozměrnému, kubickému znázornění), c) mapa čtyř proměnných (odpovídá znázornění ve čtyřrozměrném prostoru)

Pokud jde o způsob hodnotového vyjádření obsahu políčka, je zvykem, že se zapisuje pouze 1. Nevyplněný minterm se automaticky chápe jako 0. Minterm neúplně definované tabulky se většinou označuje X. Naznačená konvence je výhodná jak z hlediska přehlednosti, tak případných oprav mapy (gumování).

Cílem souřadnic mintermů pochopitelně není zavést souřadný systém mapy. Ten je pouze nutnou podmínkou k tomu, aby pomocí mapy bylo možno realizovat logické operace. Princip uvidíme jasně, prozkoumáme-li znovu pozorně všechny tři případy na obr. 7. Na všech třech nacházíme vždy některé souřadnice nebo jejich kombinace konstantní pro celý řádek nebo sloupec mapy. Vidíme, že:

a) u mapy dvou proměnných lze jednou souřadnou proměnnou definovat všechny mintermy v příslušném sloupci nebo řádku, protože všechny obsahují tutéž proměnnou ve shodném (přímém) nebo inverzním tvaru,
b) u mapy tří proměnných lze obdobným způsobem jednou proměnnou definovat všechny „podobné" mintermy jednoho řádku a dvou sloupců,
c) u mapy čtyř proměnných stačí pro obdobný popis jedna proměnná jak pro dva řádky, tak pro dva sloupce.

Tato skutečnost umožňuje při minimalizaci funkce v určitých případech chápat a používat taková políčka mapy, která vytvářejí sdružené řetězce mintermů (souvislé plochy sudého počtu mintermů, $n = 2, 4, 8$ se shodnými hodnotami 1) jako vyjádření DNT jejich společné funkce. Přitom se podle rozsahu řetězce vylučuje jedna, dvě nebo i tři proměnné a tedy i odpovídajícím způsobem zjednodušuje celý zápis funkce.

Použití mapy si ukážeme na jednotlivých praktických příkladech.

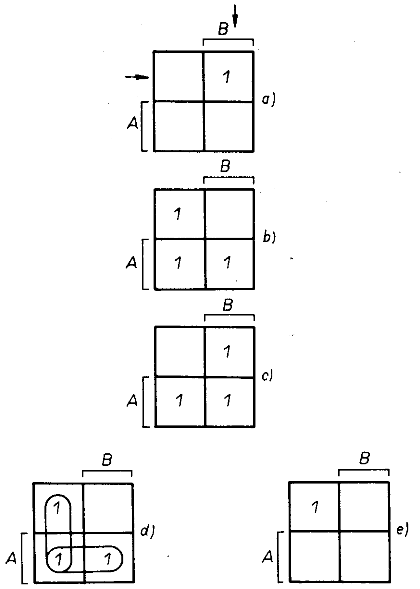
Karnaughova mapa dvou proměnných

Mapa pro dvě proměnné je pochopitelně nejjednodušší a nejpřehlednější. Využijeme ji tedy pro ukázku typických příkladů možného použití.

Zápis do mapy

Nejjednodušší je zápis mintermu, tj. jedné položky pravdivostní tabulky nebo funkce UDNT. Jako příklad zapišme minterm $\bar{A}B$, který má hodnotu 1. Po obvodu mapy vyhledáme odpovídající souřadnice $\bar{A}$, B a do políčka, které je jejich průsečíkem, zapíšeme 1. Postupným vyčerpáním všech položek pravdivostní tabulky nebo funkce UDNT zapíšeme všechna políčka mapy, která mají mít hodnotu 1. Políčka, která mají mít hodnotu 0, nezapisujeme. Příklad zápisu mintermu $\bar{A}B$ je na obr. 8a, příklad zápisu pravdivostní tabulky z obr. 5 je na obr. 8b.

Do mapy lze samozřejmě zapsat i obecnou logickou funkci ve tvaru DNT, v níž se libovolná proměnná může vyskytovat nejen jako člen mintermu, ale i v jednoduchém tvaru. V tom případě vyhledáme její jedinou souřadnici a do všech políček příslušného řádku nebo sloupce vepíšeme hodnotu 1. Vytváříme tak již zmíněné sdružené pole, řetězec mintermů. Při výskytu další proměnné v jednoduchém tvaru postupujeme stejně. Mintermy funkce DNT zapisujeme klasickým postupem. Příklad zápisu funkce $A + \bar{A}B$ je na obr. 8c.



Obr. 8. Příklady k popisu práce s mapou dvou proměnných; a) zápis mintermu $\overline{A}B$ do příslušného políčka mapy, b) zápis pravdivostní tabulky z obr. 5, je zapsána funkce $Y = AB + A\overline{B} + \overline{A}\overline{B}$, c) zápis funkce $A + \overline{A}B = A + B$, $A + B = AB + A\overline{B} + \overline{A}B$, d) minimalizace zapsané disjunkční funkce, $AB + A\overline{B} + \overline{A}\overline{B} = A + \overline{B}$, e) inverzní mapa logického součtu, $\overline{f}(A + B) = \overline{A} . \overline{B}$

Minimalizace funkce

Zapsaná funkce se v mapě dvou proměnných minimalizuje tak, že se snažíme graficky sdružovat všechna (tj. v tomto případě dvě) sousední políčka, která mají vepsánu hodnotu 1. Pokud je to možné, lze při jejich výpisu z mapy vyloučit jednu proměnnou a celá funkce se tím zjednodušuje. Políčka se sdružují tak, že sousední mintermy graficky spojujeme oválem. Jednotlivé ovály, sdružené mintermy podle určité proměnné, se dále navzájem váží, spojují.

Výraz na obr. 8a nelze minimalizovat. Sám již představuje svoji nejjednodušší formu.

Postup minimalizace funkce, zapsané z pravdivostní tabulky obr. 5 do mapy obr. 8b je na obr. 8d. Sdružením dvou dvojic sousedních mintermů $A\overline{B} + \overline{A}\overline{B}$, $A\overline{B} + AB$ získáme dvě jednoduché proměnné. Zjednodušená a v tomto případě už i minimalizovaná funkce je dána logickým součtem obou proměnných, plně určených vždy jednou souřadnicí řádku (A) nebo sloupce ($\overline{B}$). Výsledná funkce $Y = A + \overline{B}$ je shodná s výsledkem algebraického řešení, příklad (14).

Rozvoj funkce

Mnohý čtenář si možná ani nevšiml, že rozvoj funkce až do úrovně UDNT jsme již vlastně dělali. Bylo to při zápisu funkce logického součtu dvou jednoduchých proměnných A + B. Abychom tuto funkci mohli zapsat, rozložili jsme ji na jednotlivé mintermy. Můžeme jistě dobře porovnat zjednodušení, které mapa přináší ve srovnání s postupem podle Shannona. Postup při rozvoji je tedy přesně opačný, než při minimalizaci: Do všech sdružených mintermů podle řádku nebo sloupce zapíšeme 1. Rozvinutá funkce UDNT je pak rovna logickému součtu jednotlivých nalezených mintermů, viz znovu obr. 8c:

$$UDNT\ (A + B) = AB + A\overline{B} + \overline{A}B.$$

Inverze funkce

Také inverzi logické funkce, ekvivalentní algebraickému postupu, lze výhodně řešit pomocí mapy. Znamená to pouze opačně vyhodnotit, negovat hodnoty jednotlivých mintermů mapy, tedy zaměnit 0 za 1 a opačně. Pouze pro názornost je na obr. 8e znázorněna inverzní mapa logického součtu A+B z obr. 8c. Ve skutečnosti ji ovšem vůbec nekreslíme, ale pouze opačně vyhodnotíme zápis mapy v přímém tvaru, obr. 8c. Inverzní funkce logického součtu A + B je tedy

$$\overline{f}\ (AB + A\overline{B} + \overline{A}B) = \overline{A} . \overline{B},$$

což je opět možno porovnat s výsledkem, získaným např. podle Shannonova teorému.

Duální funkce

Duální funkci odvodíme jednoduše negací funkce inverzní. Tak například negací inverzní funkce logického součtu A + B z předchozího příkladu získáme duální rovnost

$$A + B = \overline{\overline{A}.\overline{B}},$$

ve které poznáváme invertovaný tvar jednoho zápisu de Morganova zákona.

I při letmém pohledu vidíme, že mapa dvou proměnných je svou jednoduchostí použití zvláště výhodná pro rychlou práci se dvěma proměnnými ve zcela obecném tvaru. Již po krátkém používání se stává nepostradatelnou.

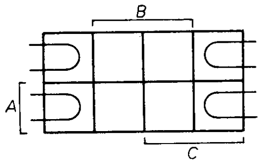
Karnaughova mapa pro tři proměnné

Všechny zásady zápisu proměnných i práce s mapou, ukázané v předchozích příkladech, platí i pro mapy a tří čtyř proměnných.

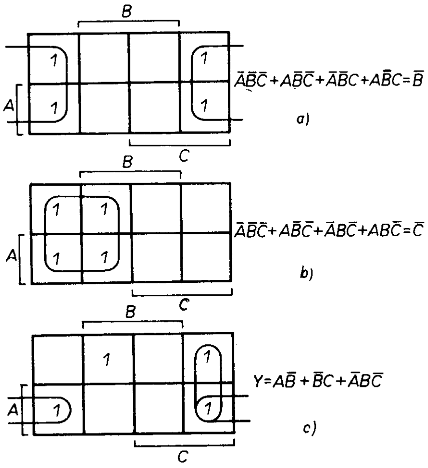
Mapa pro tři proměnné se od předchozí liší tím, že její pole je ve skutečnosti třírozměrné — jednotlivé mintermy si můžeme představit jako vrcholy krychle. Plošné znázornění všech tří typů Karnaughových map je maximálně výhodné, musí však být bráno v úvahu při hodnocení sousednosti mintermů. Na rozdíl od předchozího, plošného typu dvourozměrné mapy je u mapy tří proměnných nutno považovat za sousední mintermy i ty, které sice jako sousední zakresleny nejsou, ale leží na vzájemně protilehlých okrajích mapy. Na obr. 9 to tedy jsou políčka $\overline{A}\overline{B}\overline{C}$ — $\overline{A}\overline{B}C$ a $A\overline{B}\overline{C}$ — $A\overline{B}C$. Jejich „sousednost" ostatně ukazuje i vzájemná odlišnost inverzí jedné proměnné.

Mintermy v mapě sdružujeme do souvislých polí se dvěma nebo čtyřmi prvky. Tím můžeme z jejich sdruženého popisu vyloučit jednu nebo dvě proměnné.

Na obr. 10a je jednoduchý příklad zjednodušení funkce, jejíž všechny mintermy se nacházejí ve vzájemně protilehlých políčkách. Výsledkem minimalizace je jediná proměnná.



Obr. 9. Grafické znázornění sousednosti vnějších mintermů u mapy tří proměnných



Obr. 10. Příklady k popisu mapy tří proměnných

Obr. 10b ukazuje prakticky ekvivalentní příklad, tentokrát se sdružuje čtveřice sousedních mintermů uvnitř mapy.

Většinou ovšem tak jednoduchá řešení nenacházíme. Nakonec tedy vyřešíme běžný, praktický příklad. Máme zapsat a pomocí mapy minimalizovat funkci

$$\overline{A}\ (B \oplus C) + A\overline{B}C + A\overline{B}\overline{C},$$

v níž výraz v závorce představuje funkci výhradního (exkluzívního) součtu. Abychom mohli funkci zapsat do mapy, musíme ji upravit do tvaru DNT odstraněním závorek. Současně odstraníme operátor výhradního součtu rozpisem funkce $B \oplus C = B\overline{C} + \overline{B}C$. Potom bude mít upravená funkce tvar

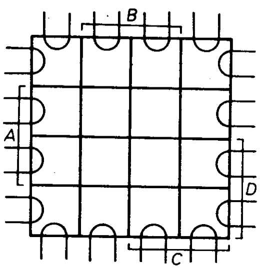
$$\overline{A}B\overline{C} + \overline{A}\overline{B}C + A\overline{B}C + A\overline{B}\overline{C}.$$

Jen pro zakímavost si všimneme, že původní funkce není zadána v optimálním tvaru. V součtu posledních dvou konjunkcí by bylo možno odstranit proměnnou C, protože $A\overline{B}C + A\overline{B}\overline{C} = A\overline{B}$. My tuto možnost klidně přehlédneme, protože při zápisu do mapy není podstatná. Zkuste si však úpravu sami.

Zápis upravené funkce do mapy i její minimalizace je na obr. 10c. Výsledná minimální funkce má tvar $\overline{B}C + A\overline{B} + \overline{A}B\overline{C}$. Vidíme, že výraznějšího zjednodušení dosáhnout nelze. Při řešení funkce pomocí logických hradel ovšem může být naznačená úprava užitečná. Lze například „vyloučit" potřebu vyjádření exkluzívního součtu úpravou minimalizovaného tvaru na výraz $\overline{B}\ (A+C) + \overline{A}B\overline{C}$, nebo $\overline{B}\ (A+C) + B\ (\overline{A+C})$ apod. Stejně užitečné může být určit inverzní funkci, ať už přímou negací minimalizované funkce, tj. $\overline{\overline{B}C + A\overline{B} + \overline{A}B\overline{C}}$, nebo pomocí inverzní mapy. Pak získáme druhý, duální tvar inverzní funkce $AB + BC + \overline{A}\overline{B}\overline{C}$. Inverzí této funkce pomocí Shannonova teorému konečně můžeme odvodit duální konjunkční tvar přímé funkce $(\overline{A}+\overline{B})$ $(\overline{B}+\overline{C})$ $(A+B+C)$. Všechny tyto možnosti mohou být v praxi užitečné, zvláště při řešení složitějších kombinačních logických sítí, kdy velmi často může být některých dílčích funkcí využito několikanásobně.

Karnaughova mapa pro čtyři proměnné

Také tato mapa je plošnou grafickou interpretací prostorového, tentokrát



Obr. 11. Znázornění sousednosti vnějších mintermů u mapy pro čtyři proměnné

čtyřrozměrného uspořádání mintermů. Proto nyní tvoří sousední mintermy nejen všechna vnitřní a rohová, ale také všechna vnější políčka na protilehlých stranách mapy, viz obr. 11. Ze souřadného systému vyplývá, že mapa pro více než čtyři proměnné již nemůže být tímto způsobem vytvořena.

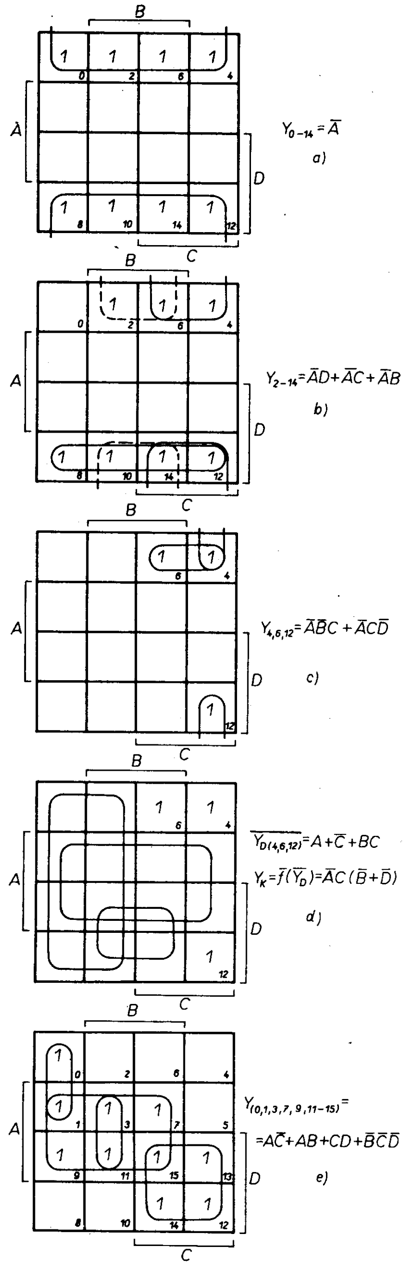
Sdružováním sousedních mintermů můžeme vytvářet souvislá pole o dvou, čtyřech nebo osmi políčkách a tak vyloučit v jejich sdruženém popisu jednu, dvě nebo tři proměnné.

Jako příklad použití mapy navrhneme disjunkční funkce pro dekodér některých vybraných kombinací sudých čísel na výstupech 4bitového binárního čítače. Příslušná pravdivostní tabulka, z níž budeme vycházet, je na obr. 12. Mapa pro dekodér všech sudých čísel včetně nuly je na obr. 13a. K postižení této funkce stačí jediná smyčka (všechna vnější políčka po obou stranách mapy). Je to logické, protože k postižení sudého nebo lichého čísla stačí kontrolovat logickou úroveň nejnižšího výstupního bitu čítače.

Číslo	Stav čítače				Y_{0-14}	Y_{2-14}	$Y_{4,6,12}$	$Y_{0,1,3,7,9,11-15}$
	D	C	B	A				
0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	0	0	1
2	0	0	1	0	1	1	0	0
3	0	0	1	1	0	0	0	1
4	0	1	0	0	1	1	1	0
5	0	1	0	1	0	0	0	0
6	0	1	1	0	1	1	1	0
7	0	1	1	1	0	0	0	1
8	1	0	0	0	1	1	0	0
9	1	0	0	1	0	0	0	1
10	1	0	1	0	1	1	0	0
11	1	0	1	1	0	0	0	1
12	1	1	0	0	1	1	1	1
13	1	1	0	1	0	0	0	1
14	1	1	1	0	1	1	0	1
15	1	1	1	1	0	0	0	1

Obr. 12. Pravdivostní tabulka připravená pro zápis několika funkcí do Karnaughovy mapy (viz obr. 13)

Mapa na obr. 13b je užita pro stanovení funkce dekodéru sudých čísel v rozsahu 2 až 14. Možností zápisu výsledné funkce je několik. Jako nejvhodnější se ukazuje naznačená varianta se třemi sdruženými poli. Ve všech konjunkcích minimalizované funkce se opět vyskytuje proměnná $\bar{A}$, definující sudé číslo. Úprava funkce na logický obvod z libovolných hradel je jednoduchá.



Obr. 13. Příklady k popisu práce s mapou čtyř proměnných

Mapa na obr. 13c je řešením funkce dekodéru tří stavů, tří čísel 4, 6, 12. Odvození výsledné funkce nepotřebuje komentáře. Na obr. 13d je pro ilustraci inverzní vyhodnocení téže funkce. Její další, Shannonovou inverzí získáme opět přímý tvar funkce, tentokrát v duálním, konjunkčním tvaru.

Konečně na obr. 13e je zápis a vyhodnocení majoritní funkce skupiny několika náhodně zvolených stavů čítače.

Existují také metody využití Karnaughovy mapy pro řešení obvodů s více než čtyřmi proměnnými. Jednu z častěji používaných nabízí princip Shannonova rozvoje. Podle (12) lze funkci s pěti proměnnými rozložit do součtu dvou funkcí, z nichž každá obsahuje čtyři proměnné a pátou je vždy konstanta, v jedné funkci I, ve druhé 0. Obě funkce se zapisují do dvou samostatných map, které se pak řeší (minimalizují) společně jako mapa jediná. Podobná řešení již ovšem vyžadují podrobnější a systematičtější studium, přičemž se v praxi využívají poměrně sporadicky.

Domníváme se, že v této kapitole jsme společně probrali hlavní praktické a jednoduché metody systematického návrhu, které mohou přispět k potlačení intuitivních metod řešení kombinačních logických obvodů. To naprosto neznamená, že zkušenost, vtip a smysl pro detail by při takové práci neměly mít místo. Naopak, většina logických sítí má ve skutečnosti celou řadu řešení, ovlivňovaných buď způsobem a postupem návrhu, volbou užitých logických obvodů nebo koncepce řešení vůbec. V současné době, kdy se na náš trh stále více dostávají různé řady univerzálních obvodů, které již zdaleka nejsou tvořeny pouze hradly NAND nebo NOR, je k návrhu co možná efektivně a optimálně řešeného obvodu třeba přistupovat s bohatším vybavením, než je pouhá znalost de Morganova zákona a principu negace.

Sekvenční obvody

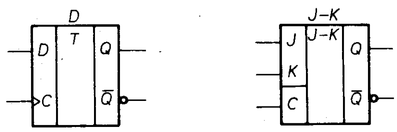
Ačkoli jsme na to výslovně neupozornili, zabývali jsme se dosud pouze řešením tzv. kombinační logiky, tj. takových funkcí, u nichž je výstupní proměnná vždy výlučně obrazem kombinace okamžitých hodnot vstupních proměnných. Čas nemá na určení funkce kombinační logiky žádný vliv.

Kombinační logika a odpovídající logické členy jsou skutečným základem číslicové techniky. Jejich samostatné využití je však značně omezené. Každá probíhající činnost (proces) se ve skutečnosti skládá z posloupnosti (sekvence) dílčích, elementárních činností, akcí. Kombinační obvod může vykonávat právě jen tuto elementární činnost. Posloupnost jednotlivých akcí umožňují definovat sekvenční obvody. Jejich funkce v sekvenční síti je však zcela odlišná. Obvodová struktura musí být řešena tak, aby se uplatnil časový rozměr, musí se používat paměťové prvky, které si nějakým způsobem pamatují předchozí, minulý stav. Jen tak lze rozhodovat, jaká akce má po právě vykonané následovat, jen tak lze definovat a řídit konkrétní proces, byť i zcela jednoduchý. Vhodnou vzájemnou součinností kombinační a sekvenční logiky lze vytvořit technické prostředky, které realizaci takového procesu umožňují.

Řekli jsme, že principem sekvenční funkce, sekvenční logiky, je uplatnění paměťového prvku v logické síti. Paměťového chování logického obvodu lze dosáhnout různými způsoby. Vždy se však jedná o nějaké uplatnění zpětné vazby, způsobující, že vlastní obvod nepůsobí v závislosti na vstupních signálech pouze směrem „ven" na výstup, ale i sám na sebe. Tím v určitém čase nabývá specifického, tzv. vnitřního stavu. Podle něj pak reaguje (změněným způsobem) na vstupní proměnné v čase následujícím. V tom pak opět nabývá nového vnitřního stavu, vytváří se nová podmínka pro reakci obvodu na vstupní proměnné atd.

V nejjednodušších sekvenčních obvodech se pro zavedení paměťového členu používají např. i běžné články *RC* (monostabilní klopný obvod). Další časté použití nachází ve stejné, tentokrát staticky definované funkci známý obvod R-S. Rozsáhlejší obvody ovšem s využitím jednoduchých obvodů R-S řešit nelze, složitost jejich návrhu by mnohonásobně překračovala únosné meze.

Aby vůbec bylo možno přistupovat k návrhu obecného sekvenčního obvodu systematickým způsobem, je nutno vycházet z principu synchronizace všech vždy vzájemně navazujících stavů. Tato synchronizace se v číslicové technice zajišťuje zavedením synchronizačního, hodinového signálu. Je většinou jednoduchý, někdy však i dvou nebo několikafázový. Celá sekvence elementárních akcí je tak periodickým vzorkováním rozdělena na stejně dlouhé (ekvidistantní) časové úseky, z nichž vždy jeden právě existující (t_n) a druhý navazující (t_{n+1}) jsou vyhodnocovány a registrovány pouze v okamžiku výskytu hodinového impulsu. Pro tyto aplikace byly postupným vývojem odvozeny od obvodu R-S některé další, dnes již standardní typy klopných obvodů. Jejich společným rysem je to, že jsou synchronní. Jejich činnost je řízena taktem hodinového signálu. Jedná se především o klopné obvody typu D, J-K a Master/Slave J-K. Mezi nimi se pak ještě rozlišují obvody se statickým a dynamickým datovým přístupem. Rozborem jednotlivých typů klopných obvodů se zabývat nebudeme. Pomíjíme je stejně jako fyzikální principy a vnitřní struktury logických hradel v předchozí kapitole. Důvodem je hlavně to, že se jedná vesměs o záležitosti dobře známé, a také skutečnost, že jsou probírány v každé příručce číslicové techniky. Kdo však má v této oblasti mezery, měl by se je snažit odstranit. Pro práci se složitějšími sekvenčními obvody je dobrá znalost funkce uvedených klopných obvodů nutná. Na obr. 14 jsou pouze pro další potřebu uvedeny obecné pravdivostní tabulky klopných obvodů D a J-K.



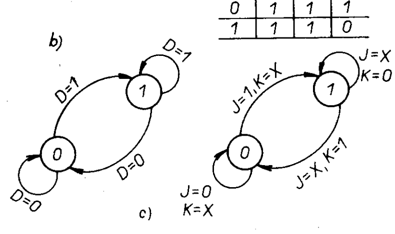
D	Q_{n+1}
0	0
1	1

a)

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

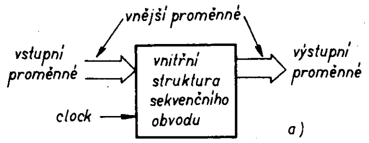
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

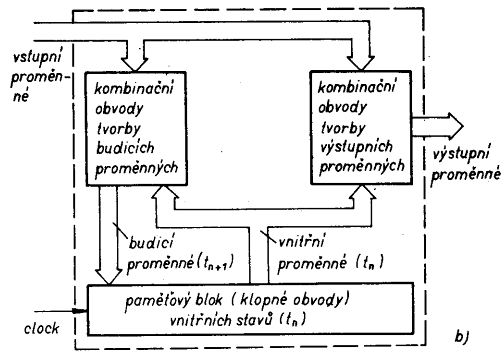
Q_n	J	K	Q_{n+1}
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	0



Obr. 14. Pravdivostní tabulky a postup odvození stavových diagramů synchronních klopných obvodů typu D a J-K; a) běžné tvary pravdivostních tabulek, b) rozepsané tvary sekvenčních funkcí pro všechny kombinace vstupních proměnných a vnitřních stavů, c) odpovídající stavové diagramy

Je jistě zcela evidentní, že návrh sekvenčních obvodů musí být ve srovnání s kombinačními obvody již z principu mnohonásobně složitější. Sekvenční obvod musí nejen v každém okamžiku požadovaným způsobem reagovat na





Obr. 15. K popisu obecného sekvenčního obvodu; a) vnitřní struktura obecného sekvenčního obvodu se navrhuje na základě analýzy vnějších proměnných, b) vnitřní struktura obecného sekvenčního obvodu s vyznačením vnitřních proměnných

aktuální logické úrovně vstupních a vnitřních proměnných (stavů), ale navíc musí vždy vytvářet nové vnitřní stavy tak, aby činnost při následujícím taktu opět za všech vnějších podmínek jednoznačně definovala požadované výstupní akce.

Odpovídající vícerozměrnou funkční strukturu obvodu nelze, při jen poněkud větší složitosti, jednoduše či přehledně definovat ani popsat dosud používanými algebraickými, tabulkovými nebo grafickými metodami. Každý dnes již jistě máme své zkušenosti například s časovým diagramem, často používaným při řešení jednoduchých obvodů. I když je časový diagram velmi užitečnou praktickou pomůckou, sám o sobě dokonalý postih obecného sekvenčního obvodu neumožňuje.

Existuje poměrně jednoduchá metoda řešení synchronizovaného sekvenčního obvodu, kterou lze doporučit pro praktické aplikace. Je založena na důsledném oddělení vnějších, již zmíněných vstupních a výstupních proměnných, které jsou při zahájení návrhu známé, od vnitřních proměnných, které je třeba vyřešit. Vzájemné relace vnějších a vnitřních proměnných jsou pak podkladem pro řešení obvodu. K tomu se přistupuje na základě odvozených budicích a konverzních funkcí.

Zatímco obr. 15a popisuje vnější proměnné, na obr. 15b je zobecněná vnitřní struktura sekvenčního obvodu. Ta se skládá zhruba ze tří bloků:
— paměťový blok slouží k definici a uchování vnitřního stavu obvodu, vzorkovaného v čase t_n,
— vstupní kombinační blok vyhodnocením relací vnějších vstupních proměnných a vnitřního stavu (t_n) nastavuje budicí funkce klopných obvodů paměťového bloku tak, aby byl definován požadovaný vnitřní stav obvodu v okamžiku příštího intervalu, vzorkovaného v čase t_{n+1},
— výstupní kombinační blok zajišťuje vyhodnocením relací ustálených (tedy synchronizovaných) vnějších vstupních proměnných a vnitřního stavu požadované stabilní úrovně výstupních proměnných po celý platný interval t_n.

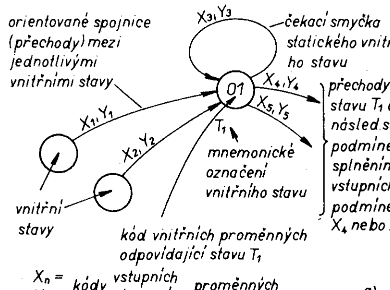
V konkrétních případech ovšem mohou některé funkční bloky nebo dokonce i některé signály v naznačené sekvenční struktuře chybět. Jsou to například oba kombinační bloky nebo vstupní proměnné. V každém případě však musí mít obvod nějakou formou vyjádřen paměťový blok a alespoň jeden vnitřní budicí signál. Existence alespoň jedné výstupní proměnné je z hlediska aktivního uplatnění sekvenčního obvodu samozřejmostí.

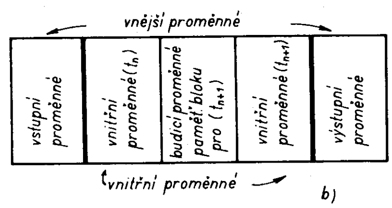
Dále uvedená metoda řešení obecného sekvenčního obvodu je kombinovaná, graficko-početní. V první fázi spočívá v interaktivním vytváření stavového diagramu a pravdivostní tabulky obvodu. Z tabulky se pak pomocí binární algebry nebo Karnaughovy mapy odvozuje minimální forma vnitřní a výstupní logiky obvodu. Metoda dokonale postihuje všechny existující kombinace vnitřních stavů obvodu a vnějších proměnných. To sice obvod dokonale popisuje za všech situací, avšak, zvláště při větším počtu stavů a proměnných, dělá jak popis, tak řešení obvodu složitým a nepřehledným.

Kombinovaná forma při zápisu řešené úlohy je velmi výhodná: graf umožňuje postihnout všechny akce, vycházející z libovolného vnitřního stavu, tabulka popisuje všechny podmínky potřebné ke každé takové akci. Ani graf, ani tabulka však nedávají obecně přímý přehled o časovém průběhu dílčích akcí, obdobný časovému diagramu. Důvody: možné větvení sledu akcí podle vstupních proměnných a chybějící časové měřítko. Proto metoda sama o sobě neumožňuje postihnout hazardní stavy (např. asynchronní čítače). Tomu se bráníme synchronizací vnějších a vnitřních proměnných nebo, někdy, odvozením dílčích časových diagramů z grafu a tabulky.

Stavový diagram a pravdivostní tabulka sekvenčního obvodu

Stavový diagram umožňuje dokonale postihnout vzájemný vztah vnějších (známých nebo požadovaných) a vnitřních (hledaných) proměnných a stavů sekvenčního obvodu (obr. 16a).





Obr. 16. Znázornění vztahů mezi stavovým diagramem (a) a pravdivostní tabulkou (b) sekvenčního obvodu; a) vztahy mezi vnitřními stavy a akcemi sekvenčního obvodu popisuje stavový diagram, b) vhodná forma pravdivostní tabulky, navazující na stavový diagram

Vnitřní stavy se v grafu označují kolečkem, jejich možné posloupnosti, představující akce obvodu, se označují šipkou orientovanými spojnicemi. Každý vnitřní stav může být buď dynamický (trvá pouze jednu periodu hodinového signálu), nebo statický. Statický vnitřní stav v grafu naznačuje čekací smyčka. Její přechodová spojnice nesměřuje k dalšímu stavu, ale vrací se zpět na stav, z něhož vychází. Z tohoto stavu může být obvod vyveden pouze změnou vstupní proměnné (nebo počáteční podmínky).

Kritickou fází návrhu stavového diagramu je správné stanovení počtu vnitřních stavů. To je záležitostí důkladné analýzy zadání úlohy, odvození potřebných relací mezi vstupními a výstupními proměnnými. Z nich vyplývá počet vnějších akcí (spojnic grafu) a tím i vnitřních stavů. Tato práce vyžaduje určitou zkušenost, kterou však lze získat velmi rychle. Každý klopný obvod (nebo paměťová buňka) je schopen vyjádřit dva vnitřní stavy (0, 1, tj. jednu vnitřní proměnnou).

Návrh sekvenčního obvodu tedy začíná analýzou úlohy a postupným sestavováním stavového diagramu. Začínáme v situaci, kdy jsme si udělali určitou představu o požadovaných vnějších akcích obvodu. K nim hledáme optimální počet a vazby vnitřních stavů jako funkce vstupních proměnných.

Prakticky souběžně s návrhem grafu je vhodné začínat s vyplňováním pravdivostní tabulky. Osvědčený příklad jejího možného uspořádání je na obr. 16b. Skládá se ze dvou hlavních polí. Pole vnějších proměnných jsou rozložena po obou okrajích tabulky. Na levé straně je pole vstupních, na pravé výstupních proměnných. Počet řádků pravdivostní tabulky je určen počtem vstupních a vnitřních proměnných obvodu. Jedna vnitřní proměnná může postihovat dva vnitřní stavy — obojí odpovídá jednomu klopnému obvodu (paměťové buňce). To znamená, má-li diagram pět vnitřních stavů, je pro jejich interpretaci v tabulce zapotřebí tří vnitřních proměnných, pro čtyři stavy dvou proměnných atd. Odtud vyplývá, že např. obvod s jednou vstupní proměnnou a čtyřmi vnitřními stavy = dvěma vnitřními proměnnými musí být v tabulce popsán $2^{1+2} = $ osmi řádky, aby byly vyčerpány všechny jejich kombinace.

Mezi vnitřními stavy diagramu a vnitřními proměnnými v tabulce musí existovat přesná vzájemná souvislost. Stejná souvislost musí být zajištěna mezi vstupními a výstupními proměnnými tabulky a spojnicemi grafu, definujícími přechody od jednoho vnitřního stavu ke druhému. Konečným cílem společné konstrukce je úplné vyplnění pravdivostní tabulky, podle které může být navržen vlastní obvod.

Nyní již k postupu návrhu. Jednotlivé stavy diagramu si označíme pomocnými, nečíselnými identifikátory. Vhodné je buď slovní, písmenové nebo kombinované (písmeno + číslice) označení. Jeden ze stavů považujeme za výchozí. Tomu stavu přísluší určité logické hodnoty vstupních a výstupních proměnných, které současně určují vazbu tohoto stavu na stav další nebo na sebe sama. Zakreslíme tedy příslušnou orientovanou spojnici a popíšeme ji těmito hodnotami. Postupným vyčerpáním všech vazeb popíšeme celý stavový diagram. Pokud se jedná o sekvenční obvod, pracující v uzavřeném cyklu, bude i stavový diagram uzavřený, cyklický. Pro jednorázové funkce musí být diagram otevřený. To se řeší tak, že poslední stav je statický a „volá sám sebe".

Následuje přiřazení kombinace všech vnitřních proměnných (které budou v další fázi využity pravdivostní tabulkou) ke každému vnitřnímu stavu diagramu. Tyto kódy, které se většinou zapisují přímo do stavového kolečka, mohou být voleny libovolně. Většinou se však snažíme, pokud je to možné, aby se buď co nejvíce blížily odpovídajícím hodnotám příslušných výstupních proměnných, nebo aby byly technicky snadno realizovatelné, např. jednoduchým čítačem nebo posuvným registrem. Tak se zjednodušují kombinační obvody.

Ve druhé fázi přepisujeme jednotlivé, vzájemně související vnější a vnitřní proměnné do jednotlivých řádků pravdivostní tabulky. Obvykle je vhodné nejprve v její levé části na jednotlivých řádcích postupně vypsat všechny možné kombinace vstupních a vnitřních proměnných a přiřadit jim požadované hodnoty proměnných výstupních. Potom do jednotlivých řádků vepíšeme i zavedené kódy vnitřních stavů (t_n). Všechny potřebné hodnoty získáme přímým přepisem ze stavového diagramu.

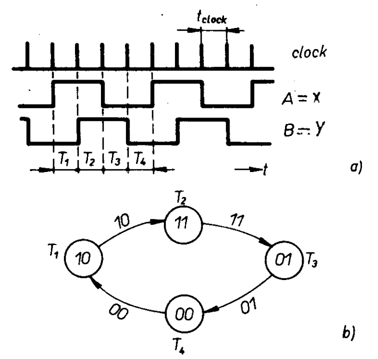
Nakonec stanovíme požadované vnitřní stavy následující fáze, t_{n+1}. I ty pohodlně zjistíme z kódového obsahu následujícího stavu, tj. toho, na který při splnění všech pdmínek v daném řádku tabulky ukazuje šipka stavového diagramu. Tím máme vyplněnu celou pravdivostní tabulku s výjimkou vnitřních sloupců, příslušejících budicím funkcím.

Ve třetí fázi je nejprve třeba odvodit všechny budicí funkce. Jejich jednotlivé řádky v tabulce musí být zaplněny vhodnými kódy, zajišťujícími změnu aktuálního vnitřního stavu (t_n, levá strana tabulky) na požadovaný budoucí stav (t_{n+1}, pravá strana). Konkrétní odvození budicích funkcí již závisí na typu užitého klopného obvodu (D, J-K). Běžným postupem při řešení kombinačního obvodu (algebra, mapa) lze odvodit minimalizovaný tvar jak budicích, tak výstupních funkcí.

Vidíme, že stavový diagram zavádí do řešení sekvenčního obvodu určitý systém. Hořejší popis není pro pochopení pracovního postupu dostatečně přehledný, byl míněn především jako poznámky k vlastní práci. Následující tři ilustrační příklady byly opět zvoleny extrémně jednoduché proto, aby řešení mohlo být sledováno bez větších problémů.

Příklady použití stavového diagramu

V prvním příkladu navrhneme generátor periodického, dvoufázového hodinového signálu s kmitočtem $f = 1/4t_{clock}$ se střídou 1:1 a s překrývajícími se fázemi při $\Delta\varphi = \pi/2$. Obvod, jehož řešení zatím považujeme za neznámé, má jediný, hodinový vstup t_{clock} a dva paralelní výstupy. Nemá tedy žádnou vstupní proměnnou!



Vnitřní proměnné (t_n)		Budicí proměnné (t_n)→(t_{n+1})		Vnitřní proměnné (t_{n+1})		Výstupní proměnné (t_n)	
A	B	D_a	D_b	A	B	X	Y
0	0	1	0	1	0	0	0
1	0	1	1	1	1	1	0
0	1	0	0	0	0	0	1
1	1	0	1	0	1	1	1

c)

Obr. 17. Časový a stavový diagram s odpovídající pravdivostní tabulkou generátoru dvoufázového hodinového signálu s fázovým překrytím

Obvyklým a v daném případě zcela opodstatněným přístupem k řešení je znázornění rozvinutého průběhu obou výstupních signálů časovým diagramem (obr. 17a). Ten především ukáže, že celá perioda se skládá ze čtyř dynamických vnějších stavů. Ukáže také, že vzhledem ke vzájemnému překrytí signálů se není třeba obávat hazardních stavů. Zkušenost může, ale také nemusí napovědět, že vhodnou základní strukturou obvodu představuje kruhový čítač nebo posuvný registr, řízený hodinovým signálem. Pro celý návrh vazeb registru a vlastně celého obvodu by stačilo analyzovat diagram jen poněkud podrobněji. Ve složitějších případech nás však podobný postup, založený na intuici, často nechá zcela na holičkách.

Proto je vhodné obvod navrhovat pomocí stavového diagramu, návrh je pak podstatně systematičnější. Z dosavadních úvah už vyplývá, že sekvenční obvod bude mít čtyři vnější výstupní stavy. Protože nemá vstupní proměnnou a každý vnější stav trvá jednu periodu t_{clock}, bude se jeho stavový diagram skládat také ze čtyř dynamických vnitřních stavů, T_1 až T_4. Protože obvod generuje periodický signál, bude mít stavový diagram jednoduchý uzavřený tvar bez větvení. Zakreslíme tedy čtyři kolečka vnitřních stavů a vzájemně je propojíme jednoduchými spojnicemi (obr. 17b). Jednotlivé stavy označíme mnemonikou, shodnou s označením časového diagramu.

Dále všem stavům přiřadíme odpovídající výstupní proměnné. Získali bychom je například rozborem zadání úlohy. Výsledek by byl stejný s tím, jaký lze zjistit z časového diagramu. Označíme-li výstupní proměnné X, Y, pak platí tabulka:

X	Y
1	0
1	1
0	1
0	0

Tyto hodnoty výstupních proměnných postupně v uvedeném pořadí dopíšeme k jednotlivým spojnicím po sobě následujících vnitřních stavů. Je lho-

stejné, kterým stavem začneme, pouze v našem případě, pro souhlas mnemonických označení vnitřních stavů v obou diagramech, začneme stavem T_1. Vstupní proměnnou obvod nemá, popis spojnic je tedy ukončen.

Přistoupíme ke kódování vnitřních proměnných. V našem případě máme mimořádnou možnost. Můžeme volit vnitřní proměnné $(A, B)_n$, shodné s výstupními proměnnými. To proto, že v celé sekvenci výstupních proměnných se žádný stav (shoda kombinací proměnných) neopakuje. Důsledkem bude možnost vyloučit výstupní, konverzní logiku. Dovnitř jednotlivých stavů diagramu tedy vepíšeme kódy, shodné s kódy na spojnicích, které z nich vystupují. Přiřadíme tedy $T_1 = 10$, $T_2 = 11$, $T_3 = 01$, $T_4 = 00$. První proměnnou vždy označíme A, druhou B. Takto je ze stavového diagramu přepíšeme do odpovídajících sloupců vnitřních proměnných $(A, B)_n$ tabulky.

Také výstupní proměnné přiřazujeme jednotlivým řádkům tabulky přepisem ze stavového diagramu. Vnitřním proměnným, zakresleným v symbolu vnitřního stavu, odpovídají výstupní proměnné, které ze stavu vycházejí. Vnitřním proměnným, zapsaným v určitém řádku pravdivostní tabulky, odpovídají výstupní proměnné na témže řádku. V tomto případě je tedy přiřazení jednoduché, protože neexistuje žádná vnější podmínka, vstupní proměnná. Proto z každého vnitřního stavu vystupuje pouze jediná spojnice. Za této situace lze vzhledem k tomu, že jsme vnitřní a výstupní stavy zvolili shodné, přímo celé pole vnitřních proměnných $(A, B)_n$ přepsat do pole výstupních proměnných $(X, Y)_n$.

Dále odvodíme vnitřní proměnné následujícího stavu (t_{n+1}). Zjistíme je vždy z následujícího stavu, na který v předchozím případě (obecně však vždy při splněné vstupní podmínce) ze stavu t_n mířila šipka.

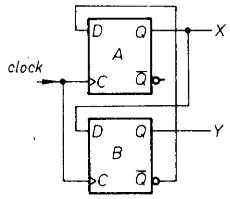
Neuškodí poznamenat, že posloupnost jednotlivých řádků v tabulce je libovolná. Důležité je pouze to, aby vzájemná korespondence mezi všemi proměnnými na témže řádku byla přesně definována. Zvláště ve složitějších případech se však projeví výhoda postupu, který jsme zvolili, tj. systematické vyčerpání všech možných kombinací vstupních (zde žádné) a vnitřních proměnných. Teprve potom doplňujeme ostatní sloupce tabulky, zásadně podle obsahu stavového diagramu.

Nyní již máme vyplněnu celou tabulku kromě budicích proměnných, které závisí na typu klopného obvodu. Protože nehrozí nebezpečí vyhodnocení hazardních stavů a všechny vnitřní stavy v diagramu mají jednoduché vstupy, nemusíme si v tomto případě s realizací klopných obvodů lámat hlavu. Použijeme jednoduché dynamické synchronní obvody typu D. Pak je odvození budicích proměnných zcela průhledné. Má-li příslušná vnitřní proměnná (např. A) ve stavu t_{n+1} nabýt hodnoty 1, píšeme do odpovídající budicí proměnné (D_a) ve stejném řádku hodnotu 1 a naopak. Tak získáme celou, kompletně vyplněnou pravdivostní tabulku. Můžeme přistoupit k minimalizaci budicích funkcí, což již umíme z předchozí kapitoly. Vzhledem k jednoduchosti použijeme algebraické řešení. Pro jednotlivé vstupy klopných obvodů vyplývá z vyhodnocení tabulky

$D_a = \overline{A}\overline{B} + A\overline{B} = \overline{B}$,
$D_b = A\overline{B} + AB = A$.

Konverzi vnitřních proměnných $(A,B)_n$ na výstupní proměnné řešit nemusíme, protože jsme volili shodné.

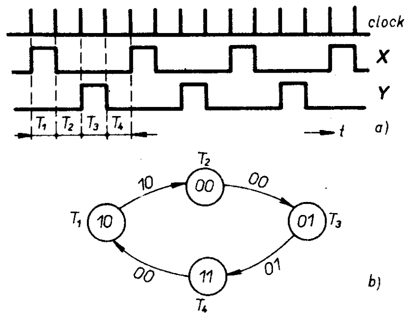
Můžeme tedy přímo nakreslit konečné schéma obvodu, plnícího zadanou funkci, obr. 18. Vidíme, že jsme jednoduchým, systematickým postupem navrhli sekvenční obvod, aniž bychom se jeho vlastní funkcí nějak zvlášť zabývali nebo využili zkušeností. Příklad byl ovšem mimořádně jednoduchý.



Obr. 18. Zapojení generátoru s fázovým překrytím výstupních signálů

Druhý příklad je velmi podobný prvnímu, pouze nepatrně obtížnější. Navrhneme opět generátor dvoufázového, tentokrát nepřekrývajícího se hodinového signálu, specifikovaného časovým diagramem na obr. 19a.

Analýzou časového diagramu zjistíme, že stejně jako v předchozím případě budou všechny výstupní stavy obvodu dynamické, ani jedna kombinace výstupních proměnných netrvá déle než jeden hodinový interval. Obvod znovu nemá žádnou vstupní proměnnou. Tomu odpovídají čtyři dynamické, mezistavové spojnice a tím i čtyři dynamické vnitřní stavy ve stavovém diagramu obvodu (obr. 19b). Proti předchozímu příkladu je zde však přece jedna odlišnost. Dvě kombinace výstupních proměnných XY=00 se v sekvenci cyklu opakují. To znemožňuje volit shodné kódování vnitřních stavů s odpovídajícími výstupními proměnnými. Ve stavovém diagramu nemohou existovat dva shodné vnitřní stavy. V tomto a celé řadě podobných případů máme v podstatě dvě možnosti. Buď rozšířit počet vnitřních stavů a tedy přidat ještě jednu vnitřní proměnnou, nebo zavést odlišné kódování vnitřních stavů od výstupních.



Vnitřní proměnné (t_n)		Budicí proměnné (t_n)→(t_{n+1})		Vnitřní proměnné (t_{n+1})		Výstupní proměnné (t_n)	
A	B	D_a	D_b	A	B	X	Y
0	0	0	1	0	1	0	0
1	0	0	0	0	0	1	0
0	1	1	1	1	1	0	1
1	1	1	0	1	0	0	0

c)

Obr. 19. Časový a stavový diagram s odpovídající pravdivostní tabulkou k příkladu návrhu dvoufázového generátoru s nepřekrývajícími se fázemi

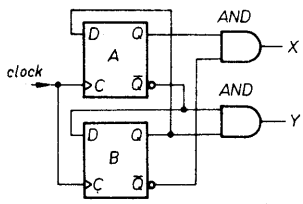
Jednodušší řešení v dané situaci přináší druhá varianta. Kódování, zavedené do diagramu a pravdivostní tabulky, se od výstupních proměnných liší jen nepatrně, pouze v jednom řádku. Proto můžeme očekávat jednoduchou výstupní, konverzní logiku. Zběžný pohled na posloupnosti vnitřních stavů t_n a t_{n+1} ukazuje, že ani v tomto případě není třeba obávat se hazardních stavů. Tabulku doplníme budicími proměnnými pro obvody typu D. Odtud, buď početně nebo pomocí mapy, odvodíme úplné budicí funkce

$D_a = \overline{A}B + AB = B$,
$D_b = \overline{A}\overline{B} + \overline{A}B = \overline{A}$.

a nakonec stejným způsobem výstupní funkce

$X = A\overline{B}$, $Y = \overline{A}B$.

Pomocí těchto funkcí již můžeme sestavit konečné schéma generátoru, obr. 20. Za povšimnutí stojí, že jádra obou generátorů, které jsme navrhovali, jsou v podstatě shodná. Je to důsledek rozdělení jejich cyklu do čtyř dynamických vnitřních stavů.

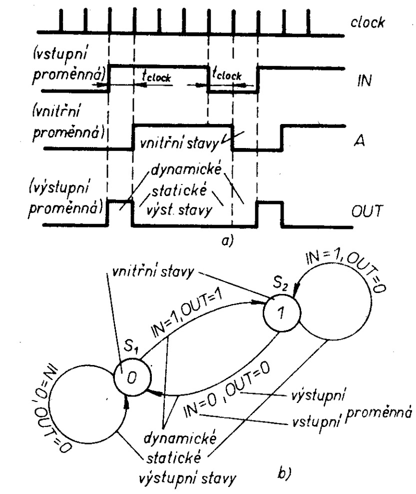


Obr. 20. Zapojení generátoru s nepřekrývajícími se fázemi

Cílem dosavadních příkladů bylo nejen ukázat možnost užití stavového grafu v netypické aplikaci, ale především znázornit co nejjednodušším, přehledným způsobem postup jeho sestavení a využití od počátku návrhu až k sestavení schématu — to bylo umožněno vyloučením vstupní proměnné.

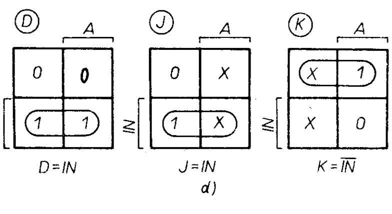
Třetí a poslední příklad již této proměnné využívá. Jedná se tedy o typický sekvenční obvod. Budeme navrhovat číslicový monostabilní obvod, resp. detektor výskytu impulsu logického signálu. Požadujeme, aby obvod generoval impuls o šířce jedné periody hodinového signálu jako reakci na každý výskyt náběžné hrany vstupního signálu. Pro jednoduchost předpokládáme, že obě hrany vstupního signálu jsou s určitým zpožděním synchronní s hodinovým taktem a jeho trvání v obou úrovních $t_{IN\ H,L} \gg t_{clock}$. Tomuto zadání odpovídají v časovém diagramu na obr. 21 stopy clock, IN (vstupní proměnná), OUT (výstupní proměnná). Rozhodujícím momentem návrhu je správné sestavení stavového grafu. V první fázi se jedná o určení počtu a typů vnitřních stavů.

Z analýzy zadání úlohy vyplývá, že jediná výstupní proměnná OUT bude mít dva stavy, jeden statický (L) a druhý dynamický (H). Tomu při jediné vstupní



Vstupní proměn. (t_n)	Vnitřní proměn. (t_n)	Budicí proměnné (t_n)–(t_{n+}) pro obvod D	obvod J-K		Vnitřní proměn. (t_{n+1})	Výstupní proměn. (t_n)
IN	A_n	D	J	K	A_{n+1}	OUT
0	0	0	0	X	0	0
0	1	0	X	1	0	0
1	0	1	1	X	1	1
1	1	1	X	0	1	0

c)



Obr. 21. Návrh synchronního detektoru náběžné hrany impulsu; a) zadání, b) stavový diagram, c) pravdivostní tabulka a odvození budicích proměnných pro řešení s klopnými obvody D i J-K (d)

proměnné IN s výlučně statickými stavovými přechody ($t_{IN} \gg t_{clock}$) vyhovuje obvod s jedinou vnitřní proměnnou A.

Tabulka tedy bude mít $2^{IN+A} = 2^2 = 4$ řádky. Do jednotlivých řádků ve sloupcích IN, A_n rozepíšeme všechny čtyři možné kombinace logických hodnot obou proměnných. Dále zakreslíme základní prvky stavového grafu s jednou vnitřní proměnnou, tj. dva vnitřní stavy S_1, S_2 a postupnou analýzou detailních požadavků na funkci obvodu odvodíme a zakreslíme potřebné vazby. Budou dvě mezistavové (dynamické) a dvě statické. Všechny vazby popíšeme odpovídající kombinací stavů vnějších proměnných (IN, OUT). Z orientace a popisu vazeb grafu přímo vyplývají proměnné OUT a A_{n+1}, které přepisem doplníme do tabulky. Správnost konstrukce grafu, přiřazení proměnných a odpovídajícího vyplnění tabulky je třeba pečlivě kontrolovat, protože jsou již přímým podkladem pro výběr vhodného typu klopného obvodu, návrh jeho budicích funkcí a výstupní kombinační logiky.

Je vhodné opakovaně obousměrně kontrolovat správnost obou zápisů, tj. správnost tabulkového zápisu kontrolujeme podle popisu stavového grafu a opačně.

Složitost vnitřní struktury řešeného sekvenčního obvodu je často velmi závislá na typech použitých klopných obvodů, (na té které pozici). Zvláště v případech s větším počtem vstupních či vnitřních proměnných se mohou dobře uplatnit i klopné obvody J-K s větším počtem součinových vstupů.

Tento příklad budeme pro názornost řešit jak s obvodem typu D, tak i J-K. Jim odpovídající budicí proměnné jsou zapsány v tabulce. Předpokládáme, že s řešením této úlohy pro obvod J-K mohou mít mnozí čtenáři problémy. Využijme příležitosti a vraťme se k obr. 14, kde jsme rozvojem běžných katalogových pravdivostních tabulek podle vstupních (D, popř. J, K) a vnitřních stavů (Q_n) ve všech jejich kombinacích odvodili příslušné vnitřní stavy Q_{n+1} i stavové grafy obou klopných obvodů. Neoznačená výstupní proměnná v grafu je totožná s vnitřní proměnnou, vepsanou do příslušného vnitřního stavu. Zatímco obvod D je zcela transparentní, obvod J-K se vzhledem k uplatnění dvou vstupních proměnných chová mezi stavy t_n a t_{n+1} rafinovaněji. Výstižnější pro jeho praktické užívání je následující tabulka, kterou lze vypsat ze stavového grafu:

Q_n	J	K	Q_{n+1}
0	0	X	0
0	1	X	1
1	X	0	1
1	X	1	0

Z tabulky vyplývá, že pro určení logické úrovně výstupu Q obvodu J-K při následujícím hodinovém impulsu se vzhledem k okamžitému vnitřnímu stavu (Q_n) vždy aktivně uplatňuje pouze jedna vstupní proměnná:

a) při přechodu z $Q_n = 0$ je to proměnná (vstup) J,

b) při přechodu z $Q_n = 1$ je to vstup K,

c) zbývající vstupní proměnná může mít vždy libovolnou úroveň,

tj. nemusí být definována.

Tímto postupem snadno vyplníme i budicí proměnné J, K v tabulce na obr. 21c. Do polí v každém řádku, která nevyžadují specifikaci logické úrovně, zapíšeme X. Tak získáme neúplně definovanou pravdivostní tabulku budicích funkcí. I když i v tomto případě je algebraické řešení jednoduché, využijeme pro ukázku práce s neúplně definovanou tabulkou mapy (obr. 21d). Nedefinovaný minterm můžeme tehdy, když to umožní minimalizaci funkce, považovat za 1.

Budicí funkce

a) pro obvod typu D: D = IN,

b) pro obvod typu J-K: J = IN, K = $\overline{IN}$.

Výstupní proměnná pro oba typy obvodů OUT = IN.$\overline{A}$. Odtud vyplývající zapojení obvodu pro obě alternativní řešení jsou na obr. 22. Na obr. 22b je také naznačená možná náhrada hradla AND hradlem NOR. (Obdobné řešení lze použít i v zapojení na obr. 20, protože i tam jsou k dispozici negované výstupy klopných obvodů.)

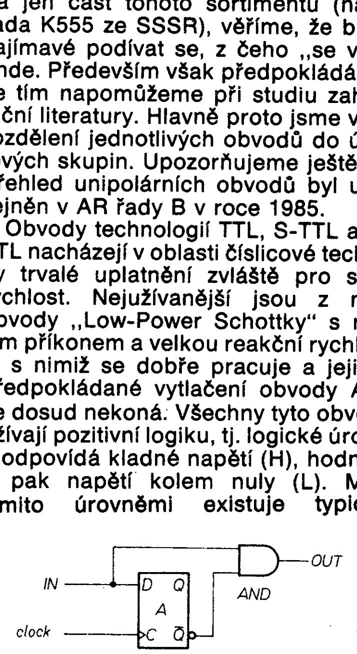
Snad se nám podařilo ukázat, že stavová analýza i syntéza sekvenčního obvodu je v praxi užitečná. Její použitelnost a přehlednost je samozřejmě, podobně jako dříve naznačené metody řešení kombinačních obvodů, omezena počtem vstupních a výstupních proměnných. To znamená, že běžné sekvenční obvody, které bývají podstatně složitější než uvedené příklady, zpravidla jako celek takovým postupem řešit nelze. Při jejich návrhu musí být úloha rozložena do kritických uzlů, ve kterých se již uplatňují také složitější obvodové celky (čítače, dekodéry, multiplexery...). Odpovídající metody řešení jsou pak nesrovnatelně obtížnější. Novou cestu návrhu takových obvodů v poslední době ukazují počítačové simulátory. I u nich se však s pojmem a významem logického stavu a stavového diagramu neustále potýkáme. Budeme se s ním stále setkávat i v jiných souvislostech.

Druhy číslicových obvodů

Kombinační a klopné obvody tvoří základ většiny složitějších obvodových celků, vytvářejících v současné době různé řady stavebnic univerzálních logických obvodů, tedy obvodů malé a střední integrace (SI, MSI), využívajících bipolární i unipolární technologie. I když se v tomto ohledu díky tuzemským výrobcům a dovozu v poslední době zlepšuje situace i u nás, zůstává faktem, že dostupný sortiment těchto obvodů je stále pouze zlomkem toho, co v jednotlivých řadách existuje na světovém trhu. Dalším stejně nepříjemným a trvajícím činitelem, který brání efektivní práci mnoha konstruktérů, je nedostatek kvalitních technických podkladů k těmto, ale hlavně ke složitějším obvodům. Každý, kdo má možnost pracovat např. s katalogy firem Texas Instruments nebo Intel potvrdí propastný rozdíl v tom, jak se o své zákazníky starají tito a jak naši výrobci. Bylo by už na čase tuto situaci, za které domácí výrobce dokáže sice zvládnout výrobu i složitého obvodu, ale k němuž si aplikátor musí shánět podklady v zahraničních materiálech, změnit.

V tomto čísle AR-B jsme se pokusili přinést stručný přehled řad TTL, Schottky a LS obvodů, vybraný z nového katalogu firmy Texas. Domníváme se, že podobný přehled řada čtenářů trvale postrádá. I když u nás je zatím dostupná jen část tohoto sortimentu (např. řada K555 ze SSSR), věříme, že bude zajímavé podívat se, z čeho „se vaří" jinde. Především však předpokládáme, že tím napomůžeme při studiu zahraniční literatury. Hlavně proto jsme volili rozdělení jednotlivých obvodů do účelových skupin. Upozorňujeme ještě, že přehled unipolárních obvodů byl uveřejněn v AR řady B v roce 1985.

Obvody technologií TTL, S-TTL a LS TTL nacházejí v oblasti číslicové techniky trvalé uplatnění zvláště pro svoji rychlost. Nejužívanější jsou z nich obvody „Low-Power Schottky" s malým příkonem a velkou reakční rychlostí, s nimiž se dobře pracuje a jejichž předpokládané vytlačení obvody ALS se dosud nekoná. Všechny tyto obvody užívají pozitivní logiku, tj. logické úrovni 1 odpovídá kladné napětí (H), hodnotě 0 pak napětí kolem nuly (L). Mezi těmito úrovněmi existuje typická



Obr. 22. Obě varianty řešení detektoru; zapojení s klopným obvodem typu D (a) a typu J-K (b)

Tab. I. Přehled obvodů TTL, Schottky a LS TTL

Hradla, buffery

Typ	TTL	S	LS	Funkce
00	X	X	X	4× 2vst. NAND, $Y = \overline{AB}$
01	X		X	4× 2vst. NAND, open
02	X	X	X	4× 2vst. NOR, $Y = \overline{A + B}$
03	X	X	X	4× 2vst. NAND, open
04	X	X	X	6× invertor, $Y = \overline{A}$
05	X	X	X	6× invertor, open
06	X		X	6× invertor/buffer, open (30 V)
07	X		X	6× buffer, open (30 V), Y = A
08	X	X	X	4× 2vst. AND, Y = AB
09	X	X	X	4× 2vst. AND, open
10	X	X	X	3× 3vst. NAND, $Y = \overline{ABC}$
11		X	X	3× 3vst. AND, Y = ABC
12	X		X	3× 3vst. NAND, open
13	X		X	2× 4vst. NAND (Schmitt), $Y = \overline{ABCD}$
14	X		X	6× invertor (Schmitt)
15		X	X	3× 3vst. AND, open
16	X		X	6× invertor/buffer, open (15 V)
17	X		X	6× buffer/driver, open (15 V)
18			X	2× 4vst. NAND (Schmitt)
19			X	6× invertor (Schmitt)
20	X	X	X	2× 4vst. NAND
21			X	2× 4vst. AND
22	X	X	X	2× 4vst. NAND, open
23	X			2× 4vst. NOR, strob. vstupy, možnost expanze $1Y = \overline{G\,(A+B+C+D) + X}$, $2Y = \overline{G\,(A+B+C+D)}$
24			X	4× 2vst. NAND (Schmitt)
25	X			2× 4vst. NOR, strob. vstup, $Y = \overline{G(A+B+C+D)}$
26	X		X	4× 2vst. NAND, open
27	X		X	3× 3vst. NOR, $Y = \overline{A + B + C}$
28	X		X	4× 2vst. NOR/buffer
30	X	X	X	1× 8vst. NAND
31			X	6× prvky hradlové zpožďovací linky
32	X	X	X	4× 2vst. OR, Y = A + B
33	X		X	4× 2vst. NAND/buffer, open
37	X	X	X	4× 2vst. NAND/buffer
38	X	X	X	4× 2vst. NAND/buffer, open
39	X			4× 2vst. NAND/buffer, open
40	X	X	X	2× 4vst. NAND/buffer, $Y = \overline{ABCD}$
50	X			2× AND-OR-INV, $Y = \overline{AB+CD+X}$
51	X	X		2× AND-OR-INV, $1Y = 2Y = \overline{AB+CD}$
51			X	2× AND-OR-INV, $1Y = \overline{ABC+DEF}$, $2Y = \overline{AB+CD}$
53	X			1× AND-OR-INV, $Y = \overline{AB+CD+EF+GH+X}$
54	X			1× AND-OR-INV, $Y = \overline{AB+CD+EF+GH}$
54			X	1× AND-OR-INV, $Y = \overline{AB+CDE+FGH+IJ}$
55			X	1× AND-OR-INV, $Y = \overline{ABCD+EFGH}$
60	X			2× 4vst. expander X = ABCD
63			X	6× interface I/U
64		X		1× AND-OR-INV, $Y = \overline{ABCD+EF+GHI+JK}$
65		X		jako 64, open
86	X	X	X	4× 2vst. EXOR, $Y = A \oplus B = A\overline{B} + \overline{A}B$
125	X		X	4× 3stavový buffer, Y = A, open při CE=H
126	X		X	4× 3stavový buffer, Y = A, open při CE=L
128	X			4× 2vst. NOR, driver 50Ω, $Y = \overline{A+B}$
132	X	X	X	4× 2vst. NAND Schmitt, $Y = \overline{AB}$
133		X		1× 13vst. NAND
134		X		1× 12vst. NAND s třístavovým výstupem
135		X		4× kombinace funkcí EXOR, 1Y až 4Y = $(A \oplus B) \oplus C$
136	X		X	4× 2vst. EXOR, $Y = A \oplus B$
260		X		2× 5vst. NOR
265	X			4× komplementární zpožďovací hradla
266			X	4× 2vst. EXNOR, open, $Y = \overline{A \oplus B} = AB + \overline{A}\overline{B}$
365	X		X	6× 1vst., 3stav. neinvertující bus driver
366	X		X	jako 365, invertující
367	X		X	(4+2)× 1vst., 3stav. neinvert. bus driver
368	X		X	jako 367, invertující
386			X	4× 2vst. EXOR
425	X			4× buffer (A, CE), 3stav., Y = A
426	X			jako 425, výstup aktivní při CE = H

Klopné obvody

Typ	TTL	S	LS	Funkce
72	X			1× J-K se vstupy AND, clock L→H, preset, clear
72	X			1× J-K Master/slave se vstupy AND, preset, clear
73	X		X	2× J-K, clock Π/TTL, H→L/LS, clear
74	X	X	X	2× D, clock L→H, preset, clear
76	X		X	2× J-K, clock Π/TTL, H→L/LS, preset, clear
78			X	2× J-K, clock H→L, preset (clock, clear)
104	X			1× Master/Slave, preset, clear, vstupy AND
105	X			1× J-K Master/Slave se vstupy AND, preset,clear
107	X		X	2× J-K, clock Π/TTL, H→L/LS, clear
109	X		X	2× J-K, clock L→H, preset, clear
110	X			1× J-K Master/Slave se vstupy AND, preset, clear
111	X			2× J-K Master/Slave, preset, clear
112		X	X	2× J-K clock H→L, preset, clear
113		X	X	2× J-K, clock H→L, preset
114		X	X	2× J-K, clock H→L, preset (clock, clear)
171			X	4× D, clock L→H,(clock, clear)
174	X	X	X	6× D, clock L→H,(clock, clear)
175	X	X	X	4× D, clock L→H,(clock, clear)
273	X		X	8× D, clock L→H,(clock, clear)
276	X			4× J-K, clock H→L,(preset, clear)
374		X	X	8× D, clock L→H, 3stav. (clock, output, control)
376	X			4× J-K L→H,(clock, clear)
377			X	8× D, clock L→H,(clock, enable)
378			X	6× D, clock L→H,(clock, enable)
379			X	4× D, clock L→H,(clock, enable)

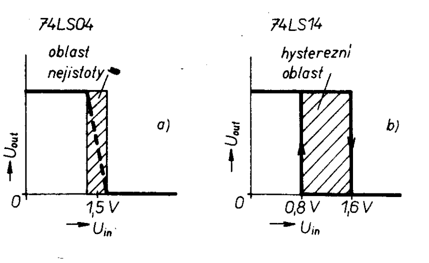
Děliče, čítače

Typ	TTL	S	LS	Funkce
56			X	kmitočtový dělič 1 : 50 (1 : 5, 1 : 5, 1 : 2)
57			X	kmitočtový dělič 1 : 60 (1 : 6, 1 : 5, 1 : 2)
68			X	2× 4bit. čítač BCD, 40 MHz
90	X		X	4× bit. dekadický čítač, 1 : 2, 1 : 5
92	X		X	4bit. dvanáctkový čítač, 1 : 2, 1 : 6
93	X		X	4bit. binární čítač, 1 : 2, 1 : 8
160	X		X	4bit. synchr. čítač BCD, clear
161	X		X	4bit. synchr. binární čítač, clear
162	X	X	X	4bit. synchr. čítač BCD, synchr., clear
163	X	X	X	4bit. synchr. binární čítač, synchr., clear
168		X	X	4bit. synchr. čítač up/down, BCD
169		X	X	4bit. synchr. čítač up/down, binární
176	X			4bit. dekadický čítač/latch s přednastavením
177	X			4bit. binární čítač/latch s přednastavením
190	X		X	4bit. synchr. čítač BCD up/down
191	X		X	4bit. synchr. čítač binární up/down
192	X		X	4bit. synchr. čítač BCD up/down, clear
193	X		X	4bit. synchr. čítač binární up/down, clear
196	X	X	X	4bit. čítač BCD/latch s přednastavením
197	X	X	X	4bit. binární čítač/latch s přednastavením
290	X		X	4bit. čítač BCD, 1 : 2, 1 : 5
292			X	digit. programovatelný dělič/timer (:2^2 až :2^{31})
293	X		X	4bit. binární čítač, 1 : 2, 1 : 8
294			X	digit. programovatelný dělič/timer (:2^2 až :2^{15})
390	X		X	2× 4bit. čítač BCD, clear
393	X		X	2× 4bit. binární čítač, clear
490	X		X	2× 4bit. čítač BCD, clear, preset
590			X	8bit. bin. čítač + výstupní registr, 3stav. výstupy
591			X	8bit. bin. čítač + výstupní registr, open
592			X	8bit. bin. čítač se vstup. registrem
593			X	8bit. čítač se vstup. reg., 3stav. port I/O
668			X	4bit. synchr. čítač BCD up/down
669			X	4bit. synchr. čítač binární up/down
690			X	4bit. synchr. čítač BCD + výstupní registr, multiplexovaný 3stav. výstup čítač/registr, clear
691			X	jako 690, binární
692			X	4bit. synchr. čítač BCD + výstupní registr, multiplex. 3stav. výstup čítač/registr, synchronní clear
693			X	jako 692, binární
696			X	4bit. synchr. čítač BCD up/down + výst. registr, multiplex. 3stav. výstupy, clear
697			X	4bit. synchr. čítač up/down binární + výst. registr, multiplex. 3stav. výstupy, clear
698			X	4bit. synchr. čítač up/down BCD + výst. registr, multiplex. 3stav. výstupy, synchronní clear
699			X	jako 698, binární

oblast nejistoty, v níž není definována ani logická úroveň, ani odpovídající chování obvodu. Opakem této běžné charakteristiky jsou stále častěji užívané obvody se Schmittovým uspořádáním vstupů, u nichž je oblast nejistoty nahrazena oblastí hystereze (obr. 23). Ve všech třech bipolárních řadách dále nacházíme tři typická, vzájemně odlišná provedení výstupního obvodu: a) klasický výstup (totem, obr. 24a) je stále aktivní, jeho výstup může mít pouze dvě úrovně binární logiky, H nebo L, b) otevřený kolektorový výstup (open) definuje pouze jednu aktivní úroveň (L), druhá výstupní úroveň (H) je podmíněna jednak zavřením výstupního tranzistoru IO, ale také vnějším obvodem, který může být společný pro výstupy několika IO. Toto provedení výstupu umožňuje realizovat

Kodéry, dekodéry, multiplexery

Typ	TTL	S	LS	Funkce
42	X		X	dekodér BCD/1 z 10
43	X		X	dekodér kódu bin +3/1 z 10
44	X			dekodér Gray +3/1 z 10
45	X			dekodér/driver BCD/1 z 10, open (30 V, 80 mA)
46	X			dekodér driver BCD/7 seg., open, aktiv. L (30 V)
47	X		X	dekodér drive BCD/7 seg., open, aktiv. L (15 V)
48	X		X	dekodér driver BCD/7 seg., aktivní H
49	X		X	dekodér driver BCD/7 seg., aktiv. H, open
137			X	dekod./demultiplexer bin/1 z 8, adresový latch
138		X	X	dekodér/demultiplexer bin/1 z 8
139		X	X	2× dekodér/demultiplexer bin/1 z 4
141	X			dekodér driver BCD/1 z 10, open (60 V)
142	X			kombin. 4bit. čítač BCD, latch, dekodér/driver BCD/1 z 10, open (60 V)
143	X			kombin. 4bit. čítač BCD, latch, dekodér/driver BCD/7 seg., 15 mA proudový výstup
144	X			jako 143, open (15 V, 25 mA)
145	X		X	dekodér driver BCD/1 z 10, open (15 V, 80 mA)
147	X		X	kodér 1 z 10/BCD
148	X		X	kodér 1 z 8/binární kód
150	X			multiplexer/selektor 1 z 16
151	X	X	X	multiplexer/selektor 1 z 8
152	X		X	multiplexer/selektor 1 z 8
153	X	X	X	2× multiplexer/selektor 1 ze 4
154	X			dekodér/demultiplexer bin/1 z 16
155	X		X	dekodér/demultiplexer bin/1 ze 4
156	X		X	2× dekodér/demultiplexer bin/1 ze 4, open
157	X	X	X	4× multiplexer/selektor 1 ze 2, neinvertující
158	X	X	X	4× multiplexer/selektor 1 ze 2, invertující
159	X			dekodér/demultiplexer bin/1 z 16, open
184	X			konvertor kódu BCD/binární
185	X			konvertor binární kód/BCD
246	X		X	dekodér driver BCD/7 seg., aktiv. L, open (30 V)
247	X		X	dekodér driver BCD/7 seg., aktiv. L, open (15 V)
248	X		X	dekodér driver BCD/7 seg., aktiv. H
249	X		X	dekodér/driver BCD/7 seg., aktiv. H, open
251	X	X	X	multiplexer/selektor 1 z 8, kompl. 3stav. výstup
253			X	2× multiplexer/selektor 1 ze 4, 3stav. výstupy
257		X	X	4× multiplexer/selektor 1 ze 2, neinvert. 3stav. výstup
258		X	X	4× multiplexer/selektor 1 ze 2 invert. 3stav. výst.
298	X		X	4× 2vst. multiplexer s výstupním registrem, (clock)
347			X	dekodér driver BCD/7 seg., open (7 V)
348			X	prioritní kodér 1 z 8/bin, 3stav. výstupy
351	X			2× multiplexer/selektor 1 z 8, 3stav. výstupy, invert.
352			X	2× multiplexer/selektor 1 z 4, invertující
353			X	2× multiplexer/selektor 1 z 4, invert. 3stav. výstupy
354			X	8bit vst. latch+multiplexer 1 z 8, kompl. 3stav. výst.
355			X	8bit vst. latch+multiplexer 1 z 8, komplement. open
356			X	8bit vst. registr+multiplexer 1 z 8, kompl. 3st. výst.
357			X	8bit vst. registr+multiplexer 1 z 8, komplement. open
398			X	4× 2vst. multiplexer+výst. registr, komplement. výst.
399			X	4× 2vst. multiplexer+výst. registr, jednoduché výst.
445			X	dekodér driver BCD/1 z 10, open (7 V)
447			X	dekodér driver BCD/7 seg., open (7 V)
604			X	8× 2vst. multiplexer, vstup. latch, 3stav.
605			X	8× 2vst. multiplexer, vstup. latch, open
606			X	jako 604, na úkor rychlosti eliminované hazardní stavy
607			X	jako 605, na úkor rychlosti eliminované hazardní stavy



Obr. 23. Stylizované průběhy převodních charakteristik hradel s klasickým (a) a Schmittovým uspořádáním vstupních obvodů (b)

Registry

Typ	TTL	S	LS	Funkce
91	X		X	8bit. posuv. reg., sériový IN, sériový OUT
94	X			4bit. posuv. reg., paralel. IN, sériový OUT
95	X		X	4bit. posuv. reg., paralel. IN, OUT, posuv vpřed/vzad
96	X		X	5bit. posuv. reg., sériový ↔ paralelní IN, OUT
164	X		X	8bit. posuv. reg., sériový IN, paralelní OUT
165	X		X	8bit. posuv. reg., paralelní IN, sériový OUT
166	X		X	8bit. posuv. reg., paralelní IN, sériový OUT
199	X			8bit. posuv. reg., paralelní IN, paralelní OUT, posuv vpřed/vzad
295			X	4bit. posuv. reg., sériový ↔ paralelní IN, OUT, posuv vpřed/vzad, 3stav. výstupy
299		X	X	8bit. obousměrný univerz. posuv./paměť. registr, 3stav. výstupy
322			X	8bit. posuv. reg., s multiplexovanými 3stavovým portem IN/OUT, znaménkové rozšíření
323			X	8bit. obousměrný posuv./paměť. reg., 3stav. port
173	X		X	4bit. D-registr, 3stav. výstupy
178	X			4bit. posuv. reg., paralel. IN, OUT
179	X			4bit. posuv. reg., paralel. IN, OUT, clear
194	X	X	X	4bit. obousměr. posuv. reg., paralelní IN, OUT
195	X	X	X	4bit. posuv. reg., sériový ↔ paralelní IN, OUT
198	X			8bit. obousměr. posuv. reg., paralelní IN, OUT
395			X	4bit. posuv. reg., sériový ↔ paralelní IN, OUT, 3stav.
396			X	2× 4bit. D-registr
594			X	8bit. sériový IN, paralel. OUT posuvný registr s výstupním paralelním registrem typu D
595			X	jako 594, 3stav. výstupy, nemá clear výst. reg.
596			X	jako 595, open
597			X	8bit. paralel. IN, sériový OUT posuv. registr se vst. paralel. registrem
598			X	8bit. posuv. reg. se vstupním paralel. registrem, paralelní 3stav. port I/O
599			X	jako 594, open
671			X	4bit. posuvný/paměťový 3stav. reg., clear
672			X	jako 671, synchr. clear posuv. registru
673			X	16bit. posuv. IN/OUT + 16bit. paralel. výst. reg.
674			X	posuv. reg. 16bit. paralel. IN, sériový OUT

Vysílače, přijímače, budiče sběrnice

Typ	TTL	S	LS	Funkce
140		X		2× 4vst. NAND driver 50 Ω
226		X		4bit. paralelní latch/transceiver, 3stav. výst.
240		X	X	2× 4bit. driver/receiver, 3stav. výstupy, invert.
241		X	X	jako 240, neinvertující
242			X	4bit. transceiver, invert. 3stav. výstupy
243			X	jako 242, neinvertující
244		X	X	2× 4bit. driver/receiver, neinvert. 3stav. výst.
245			X	8bit. transceiver, neinvert. 3stav. výstupy
436		X		6bit. TTL/MOS interface/driver
437		X		6bit. TTL/MOS interface/driver
440			X	3cestný 4bit. transceiver, neinvert., open
441			X	jako 440, invert., open
442			X	jako 440, neinvert., 3stav.
443			X	jako 440, invert., 3stav.
444			X	jako 440, invert./neinvert., 3stav.
448			X	jako 440, invert./neinvert., open
446			X	4bit. transceiver, invert., individ. řízení směru
449			X	jako 446, neinvertující
540			X	8bit. invert. buffer/driver, 3stav.
541			X	jako 540, neinvertující
620			X	8bit. transceiver, invert., 3stav.
621			X	jako 620, neinvert., open
622			X	jako 620, invert., open
623			X	jako 620, neinvert., 3stav.
638			X	8bit. transceiver, invert., A bus-open, B bus-3stav.
639			X	dtto, neinvertující
640			X	8bit. transceiver, invert., 3stav.
641			X	jako 640, neinvert., open
642			X	jako 640, invert., open
645			X	jako 640, neinvert., 3stav.
643			X	8bit. transceiver, invert./neinvert., 3stav.
644			X	jako 643, open
646			X	8bit. transceiver/registr, neinvert., 3stav.
647			X	jako 646, neinvert., open
648			X	jako 646, invert., 3stav.
649			X	jako 646, invert., open

tzv. montážní (wired) logický součin (AND) — na úrovni společného signálu se podle této funkce podílejí všechny zúčastněné obvody (obr. 24b), c) třístavový výstup (tri-state) je charakteristický třemi výstupními stavy — dvěma aktivními (binárními H, L) a třetím pasívním, s velkou impedancí, odpovídajícím v praxi „izolaci" výstupního obvodu (obr. 24c). Do tohoto stavu je

Latche

Typ	TTL	S	LS	Funkce
75	X		X	4bit. latch s komplement. výstupy
77	X		X	4bit. latch
100	X			2× 4bit. latch
116	X			2× 4bit. latch
118	X			6× latch R-S
119	X			6× latch R-S, rozšířené vstupy
259	X		X	8× Ibit. adres. latch
279	X		X	4× latch R-S
373		X	X	8bit. latch D, 3stav. výstupy
375			X	4bit. latch, komplement. výstupy

Komparátory

Typ	TTL	S	LS	Funkce
85	X	X	X	4bit. komparátor bin/BCD, výstupy >, <,=
682			X	8bit. komparátor bin/BCD, výstupy >,=
683			X	jako 682, open
686			X	8bit. komparátor, výstupy >,=, output enable
687			X	jako 686, open
688			X	8bit. komparátor, výstup =, output enable
689			X	jako 688, open

Multivibrátory

Typ	TTL	S	LS	Funkce
121	X			monostabilní multivibrátor
122	X		X	znovuspustitelný monostabilní obvod, clear
123	X		X	2× znovuspustitelný mono, clear
221	X		X	2× monostabilní multivibrátor (Schmitt trigger vstupy)
422			X	znovuspustitelný monostabilní obvod
423			X	2× znovuspustitelný monostabilní obvod

Oscilátory

Typ	TTL	S	LS	Funkce
124		X		2× napěťově řízený oscilátor (VCO)
320			X	krystalem řízený oscilátor
321			X	krystalem řízený oscilátor
624			X	VCO, komplement. výstupy, vstupy enable, range
625			X	2× VCO, komplement. výstupy
626			X	2× VCO, komplement. výstupy, enable
627			X	2× VCO, jednoduché výstupy
628			X	VCO, komplement, výstup, enable, range, ext. teplot. komp.
629			X	2× VCO, jednoduché výstupy, enable, range

Paměti typu FIFO

Typ	TTL	S	LS	Funkce
222			X	6× 4bit. asynchr. FIFO, enable IN, OUT, 3stav.
224			X	16× 4bit. asynchr. FIFO, 3stav.
227			X	16× 4bit. asynchr. FIFO, enable IN, OUT, open
228			X	16× 4bit. asynchr. FIFO, open
225		X		16× 5bit. asynchr. FIFO, 3stav.

Aritmetika

Typ	TTL	S	LS	Funkce
80	X			1bitová úplná binární sčítačka
82	X			2bit. úplná binární sčítačka
83	X		X	4bit. úplná binární sčítačka
97	X			synchronní 6bit. poměr. binární násobička
167	X			synchronní 4bit. poměr. dekadická násobička
181	X	X	X	4bit. ALU/funkční generátor
182	X	X		předpovědní generátor carry pro bin. sčítačky
183			X	2× 1bit. úplná sčítačka (carry-save)
261			X	2bit. × 4bit. paralel. binární násobička
274		X		4bit. × 4bit. binární násobička, 3stav. výst.
275		X	X	7bit. řezový Wallace tree
281		X		úplný 4bit. binární akumulátor
283	X	X	X	4bit. úplná binární sčítačka
284	X			4bit. × 4bit. paralelní bin. sčítačka
285	X			jako 284
381		X	X	ALU/funkční generátor
382		X	X	jako 381
384			X	8bit. × 1bit. násobička v 2's doplňku
385			X	4× sériová sčítačka/odčítačka
681			X	4bit. paralelní binární akumulátor

Chybové detektory

Typ	TTL	S	LS	Funkce
630			X	16bit. paralelní chybový detektor, 3stav.
631			X	jako 630, open
636			X	8bit. chybový detektor, 3stav.
637			X	jako 636, open

Mapovací obvody

Typ	TTL	S	LS	Funkce
610			X	expender/mapovač adresové sběrnice CPU (*n* bitů) na systémovou (*n* +8bitů), výst. latch, 3 stav.
611			X	jako 610, výst. latch, open
612			X	jako 610, 3stav.
613			X	jako 610, open

Kontroléry dynamických RAM

Typ	TTL	S	LS	Funkce
600			X	transparentní/burst refresh kontroler pro 4/16k RAM
601			X	transparentní/burst refresh kontroler pro 64k
602			X	cycle steal/burst refresh kontroler pro 4/16k RAM
603			X	cycle steal/burst refresh kontroler pro 64k
608			X	řadič paměťových cyklů: read, write, read — modify — write, RAS — only refresh

Různé

Typ	TTL	S	LS	Funkce
120		X		2× impulsní synchronizátor/driver
170	X		X	4× 4bit. registrový zápisník, read, write, open
172	X			16bit. registr. pole (8× 2 bity) s nezávislým adresováním
180	X			9bit. paritní generátor
278	X			4bit. prioritní registr se vstup. latchem
280		X	X	9bit. paritní generátor
297			X	digitální PLL
412		X		8bit. univerzální latch (ekvivalent 3212)
428		X		kontroler + bus driver (ekvivalent 8228)
438		X		kontroler + bus driver (ekvivalent 8238)
670			X	4× 4bit. registr. zápisník, 3stav. výst.
2000			X	směrový diskriminátor + 16bit. čítač up/down + výst. registr pro inkrementální snímače

obvod přiváděn speciálním řídicím signálem. Třístavové řešení výstupního obvodu umožňuje (vzhledem k obvodům s otevřeným kolektorem) dosáhnout větší přenosové rychlosti, ale především, využitím několika třístavových, vhodně řízených obvodů, umožňuje řízený, obousměrný simplexní přenos logických signálů mezi libovolným počtem účastníků na jednom vedení. Řízení musí být zajištěno tak, aby vždy pouze jeden z účastníků byl aktivní, tj. vysílal na vedení. Jinak by „se střetly" dva nebo několik aktivních výstupních obvodů s nedefinovanými úrovněmi.

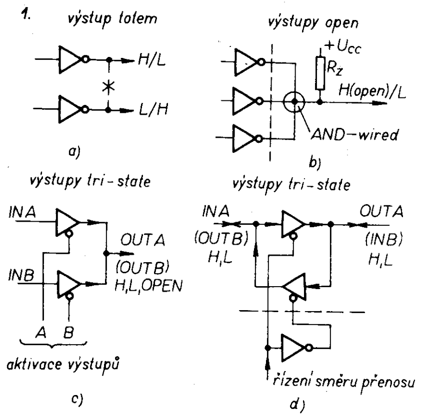
Typický vstupní proud hradla LS TTL je I_{IL} = 400 μA, zpoždění hradla t_{PLH}, t_{PHL} je typicky menší než 15 ns. Zatížitelnost výstupu standardního hradla LS je pro řadu 54 typicky 10 vstupů (I_{OL} = 4 mA), pro řadu 74 pak 20 vstupů (I_{OL} = 8 mA).

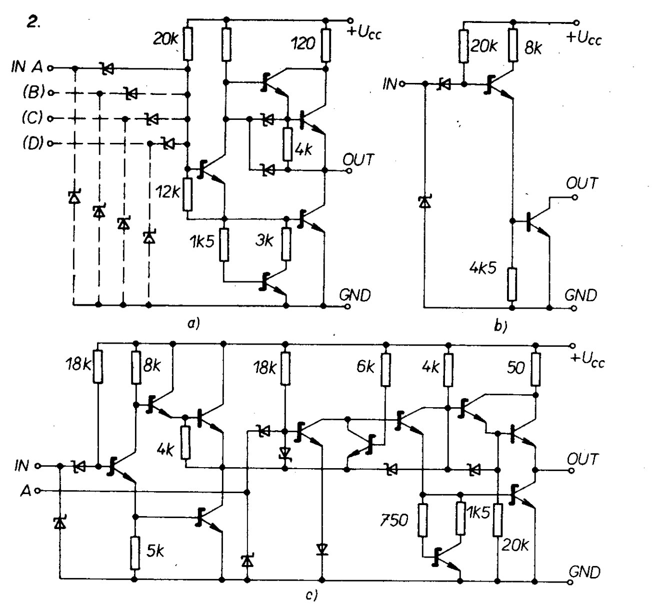
Zmíněný přehled všech tří základních řad bipolárních obvodů je v tab. I. Ve stručných poznámkách jsme většinou záměrně volili anglické označení, a to především tehdy, když jsme se domnívali, že jeho český ekvivalent není vždy přesně chápán nebo zasluhuje bližšího vysvětlení. Systematické rozčlenění tabulky je dobrou příležitostí k tomu, abychom si stručně popsali hlavní standardní funkční bloky, z nichž se každý číslicový systém skládá, nebo které v něm mohou být použity.

Hradla

První blok tabulky tvoří logická hradla (gates). V jednotlivých řadách jsou obsaženy všechny základní, součtové a součinové funkce jak v přímém, tak negovaném tvaru až do úrovně čtyř vstupních proměnných, s klasickým i otevřeným výstupem. Dále jsou zde silně zastoupeny obvody typu AND-OR-INVERT s různou úpravou funkcí vstupů a obvody typu EXOR, EXNOR. Při tak širokém sortimentu mohou být logické funkce optimalizovány skutečně účinně.

Pojmem buffer se obecně označuje





Obr. 24. 1 — základní využití hradel se standardními, open a třístavovými výstupy; a) hradla se standardními výstupy nemohou pracovat do společné zátěže, neboť by se střetávaly opačné výstupní úrovně (a přetěžovaly výstupy IO), b) realizace montážního součinu invertory nebo hradly s otevřenými kolektorovými výstupy, c) sdílení společné výstupní sběrnice s třístavovými hradly, d) realizace obousměrného řízeného přenosu s třístavovými hradly;
2 — vnitřní struktury základních typů hradel LS TTL; a) standard NAND (00, 04, 10, 20, 30), b) „open" invertor (05), c) tri-state (třístav., 125)

stykový, oddělovací stupeň. Buffer tedy odděluje logické signály s různou napěťovou nebo výkonovou úrovní.

Driver v popisu hradla znamená označení jeho akční funkce, zpravidla je to člen s výrazně výkonovým charakterem výstupu.

Jednotlivá označení bývají různě kombinována, aby výsledná specifikace byla co nejvýstižnější.

Klopné obvody

Ve skupině klopných obvodů (flip-flops) nacházíme bohatý výběr dynamických obvodů D a J-K různého provedení co do funkce, počtu instalovaných obvodů a dostupných vývodů synchronních i asynchronních vstupů a výstupů. V kompromisu s možným počtem vývodů pouzdra IO tak vznikla „prefabrikovaná" řada obvodů, dobře pokrývající potřebu volby jednotlivých typů pro konkrétní aplikaci.

Čítače

Skupina čítačů (counters) nabízí výběr z několika desítek různých čítačů (synchronních, asynchronních, obousměrných, přednastavitelných, binárních nebo BCD), v některých případech doplněných vstupním nebo výstupním registrem/latchem.

Kodéry /dekodéry, multiplexery/ demultiplexery

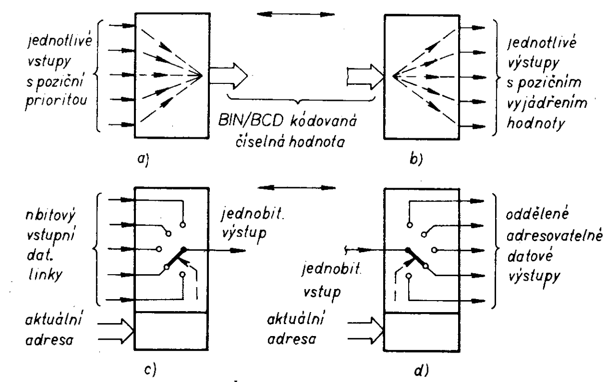
Kodér (prioritní kodér je kombinační obvod, který kóduje, převádí logické úrovně jednotlivých z řady vstupních signálů, z nichž každý má určitou pevně přiřazenou prioritu, do některého číselného kódu (bin, BCD...). Výstup udává hodnotou tohoto kódu pozici vstupního signálu, který má ze všech signálů v aktivní logické úrovni nejvyšší prioritu.

Dekodér je kombinační obvod, jehož funkce může být považována za opak předchozí. Dekodér převádí číselnou hodnotu z jednoho kódu do druhého. Počet logických výstupů dekodéru může být obecně 1 až *n*.

Multiplexer je obvod s oddělenými datovými a adresovými vstupy a zpravidla jedním výstupem. Podle obsahu aktuální, právě nastavené adresy multiplexer vybírá jednu ze vstupních proměnných a přiřadí ji (v přímém nebo inverzním tvaru) na výstup.

Demultiplexer opět pracuje obráceně. Má jeden vstup, adresové pole a několik oddělených datových výstupů. Okamžitý stav „jediné vstupní proměnné" se podle aktuální adresy přepíná na jediný, právě adresovaný výstup.

Schématické znázornění funkce jednotlivých obvodů této skupiny je na obr. 25.

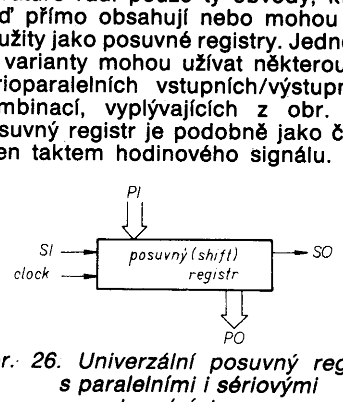


Obr. 25. Prioritní kodér/dekodér a multiplexer/demultiplexer představují vzájemně opačné, doplňkové funkční bloky; a) prioritní kodér, b) dekodér, c) multiplexer, d) demultiplexer

Registry

Do této skupiny se v zahraniční literatuře řadí pouze ty obvody, které buď přímo obsahují nebo mohou být použity jako posuvné registry. Jednotlivé varianty mohou užívat některou ze sérioparalelních vstupních/výstupních kombinací, vyplývajících z obr. 26. Posuvný registr je podobně jako čítač řízen taktem hodinového signálu. Posuvy a rotacemi vpřed (vpravo) a vzad (vlevo) lze posouvat, vysouvat a rotovat jednotlivé bity i celý obsah registru oběma směry. Tak se testují jednotlivé bity a vytvářejí základní funkce aritmetických a logických operací. Tyto obvody jsou v číslicové technice nezastupitelné.

V příslušné části tabulky vidíme, že jednotlivé typy registrů se liší bitovou kapacitou, různými možnostmi využití vstupních/výstupních funkcí a řízení. U některých typů nacházíme vstupní nebo výstupní paralelní registry pro přechodné zachycení zpracovávaných dat v paralelním tvaru.

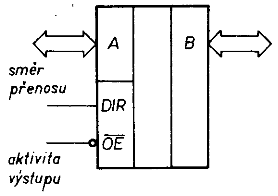


Obr. 26. Univerzální posuvný registr s paralelními i sériovými vstupy/výstupy

Vysílače/přijímače, budiče sběrnic

Tato skupina obvodů tvoří základ řízené komunikace mezi dvěma a větším počtem logických bloků. Jejich funkce je založena na principu třístavové logiky a společné sběrnice signálové cesty. Obousměrná, vždy však pouze jednocestná komunikace s jediným aktivním zdrojem signálu tedy odpovídá obr. 24 s tím rozdílem, že se neuskutečňuje po jediném vodiči, ale po celé *n*bitové sběrnici, obr. 27.

Vysílač/přijímač, označovaný též jako transceiver, je tedy kombinační obvod, realizující tuto funkci pro jedno stykové místo v *n*bitové šíři. Směr přenosu (direction) se ovládá jedním, aktivita přenosu druhým (enable) řídicím signálem. Přitom bývá většinou alespoň jedna z cest řešena jako výkonová. Jsou-li jako výkonové reali-



Obr. 27. Obousměrný budič sběrnice

zovány obě přenosové cesty, obvody se chovají jako obousměrné výkonové budiče sběrnic.

Průchozí registry

Funkci tohoto typu registrů, tvořených klopnými obvody se statickým přístupem, nejlépe charakterizuje jejich původní anglické označení latch (zábrana, závora). Obvod je aktivován ne hranou, ale úrovní řídicího signálu. Při jeho aktivní úrovni je obvod průchozí, chová se jako běžné hradlo. S týlovou hranou ukončeného řídicího signálu jsou v registru zachycena jako platná ta data, která se na vstupu registru vyskytovala právě v tomto okamžiku.

Komparátory

Hodnotové (magnitude) komparátory se užívají k porovnání dvou čísel binárních nebo BCD v absolutním kódu, bez znaménka. Komparátor je opět kombinační obvod. Jeho výstupem jsou oddělené jednobitové výstupy výsledku porovnání >,< a =, nebo pouze některé z nich.

Multivibrátory

Tato skupina napěťové a teplotně relativně velmi stabilních obvodů je dobře známá. Snad stojí za poznámku, že běžný multivibrátor v monostabilním zapojení nemůže být opakovaně spuštěn před ukončením právě generovaného výstupního impulsu. To naopak umožňují „znovuspustitelné“ (retriggerable) obvody. Ty je možno startovat kdykoli, i opakovaně, čímž se odpovídajícím způsobem prodlužuje výstupní impuls.

Použití multivibrátorů *RC* v číslicových systémech by mělo být omezeno na nejnutnější míru a směrováno pouze do nekritických oblastí.

Oscilátory

Zde nacházíme obvody krystalem (XCO) a napětím (VCO) řízených oscilátorů s různými variantami řídicí sekce a výstupního obvodu, umožňující pokrýt široký kmitočtový rozsah a PLL.

Aritmetické obvody

Tuto skupinu tvoří *n*bitové binární sčítačky a násobičky, aritmetickologické jednotky a různé dílčí nebo pomocné obvody, nezbytné k realizaci konkrétních účelových sestav, vytvářených některými z uvedených obvodů. Principy těchto obvodů se budeme zabývat dále.

Chybové detektory

Tyto detektory generují na principu modifikovaného Hammingova kódu kontrolní slovo *n*bitového vstupního signálu.

Mapovací obvody

Tyto zajímavé obvody (memory mappers) jsou určeny k rozšíření adresovací schopnosti mikroprocesoru. Vytvářením tzv. mapovacích registrů umožňují realizovat stránkové adresování paměti.

Řadiče dynamických pamětí

Obsahují obvody (synchronní čítače, buffer/drivery, časovače *RC* a doplňková logika) zajišťující oživování (refresh) dynamických pamětí RAM.

Paměti FIFO

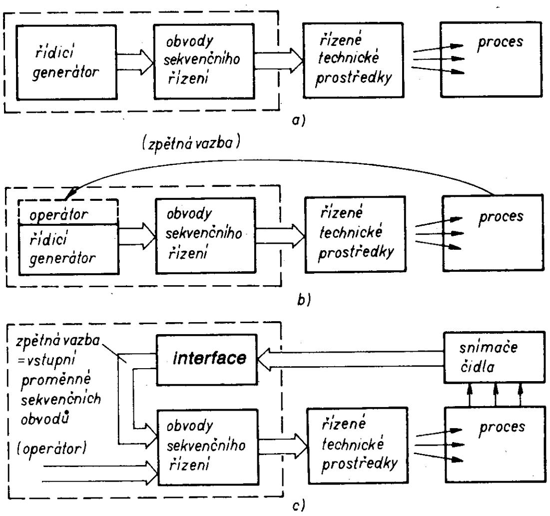
Jsou to rychlé, stavebnicově rozšiřitelné (co do šířky slova i kapacity) statické paměti LS typu first in — first out (první dovnitř, první ven). Rychlosti zápisu a čtení jsou vzájemně nezávislé.

Použití sekvenčních obvodů (automatů)

V předchozích kapitolách jsme si již do řešení klasických logických obvodů vnesli určitý pořádek. Získali jsme názor na obecný sekvenční obvod, pracující v reálném čase, popsaný vstupními a výstupními proměnnými. Tyto veličiny jsme však doposud chápali jako nediskutovatelné parametry, dané požadavky ze strany vnějšího prostředí.

Všimněme si nyní skutečných problémů, které při vazbě logického systému na vnější prostředí vyvstávají. Uvážíme-li postupně všechny možnosti, dostáváme se k vazbám, znázorněným na obr. 28. Na obr. 28a je zachycena situace, kdy obvod ovládá řízené prostředky, tedy jednotlivé akční

Obr. 28. Tři základní možnosti vazby obecného sekvenčního obvodu na vnější prostředí; a) invariabilní sekvenční automat, b) operátor má možnost vlivu na průběh sekvence, c) zpětnovazební smyčka umožňuje zavést reakci systému na reálný stav procesoru



členy na základě vstupních proměnných. Tyto parametry mohou být zadávány buď automaticky, např. nadřazeným sekvenčním systémem, nebo ručně, operátorem. V prvním případě je tedy celý proces řízen pevným neměnným programem. To však přináší celou řadu nedostatků, pro které takový systém často nemůže být použit. Uvažujme jednoduchý příklad, řízení dopravy materiálu ve skladu nebo na výrobním pásu. Jakmile dojde k nepředvídaným situacím (časovým zpožděním, nedostatku určitého materiálu, skladovací kapacity...), nastává havarijní stav. Možné řešení problému znázorňuje obr. 28b. Zde je jednoduchý řídicí systém nahrazen operátorem. Ten opět systému zadává vstupní proměnné, které jsou vnitřní sekvenční funkcí rozvíjeny do složitějších posloupností jednotlivých výstupních akcí. Operátor však nyní do procesu řízení zavádí, díky vlastní inteligenci, zpětnou vazbu. Celý systém řídí na základě vyhodnocování skutečného stavu procesu v reálném čase. Výhodnější je zřejmě kombinace obou metod — automatické řízení procesu s dohledem operátora, který by zasahoval jen v případě kritických situací. I tak je do řízení zaváděna druhá stránka věci — tzv. lidský faktor, který může být zdrojem chyb a omylů.

Příklad uspořádání skutečného zpětnovazebního řízení obvodu je na obr. 28c. Flexibilita procesu, zaváděná v předchozím případě využitím schopností operátora, je zde technicky reálně vyjádřena zpětnovazební cestou akce ↔řízení. To obecně vyžaduje nasadit promyšlený systém vhodných čidel (pro náš případ koncové spínače, počítadla, indikátory a čítače stavů, množství...) a převodníků, snímajících reálný stav. I v tomto uspořádání je možná součinnost operátora.

Přes letmé naznačení problémů, vyskytujících se při realizaci obou typických příkladů, je jistě dobře patrné, jak náročné požadavky jsou v praxi kladeny na celý sekvenční systém. Přitom nelze přehlédnout, že již samo nalezení vhodného způsobu řešení úlohy je často velmi složité. Dopouštíme se při něm zákonitě řady chyb, které mnohdy mohou být nalezeny a opravovány až v průběhu obvodového řešení, které tak může být zcela zvráceno. Rovněž možnosti úprav v řízení procesu na základě pozdějších požadavků jsou při klasickém, „hardwarovém“ naprogramování funkce systému někdy velmi omezené, někdy nemožné. Sekvenční obvody se stávají rozsáhlými, se složitými několikaúrovňovými hierarchickými vazbami. Je nutné brát v úvahu i požadavky na prioritní zpracování některých vnějších událostí, havarijních stavů atd. Vyrovnat se se všemi těmito i jinými problémy (doba realizace, náklady, spolehlivost...) již klasická elektronika nedokáže. Novou cestu již dávno ukazovala výpočetní technika. Praktického využití v širokém měřítku se jí však dostává až nyní, díky radikálnímu zlepšení technických i ekonomických parametrů všech potřebných složek o několik řádů a jejich masové dostupnosti.

Paměťové obvody

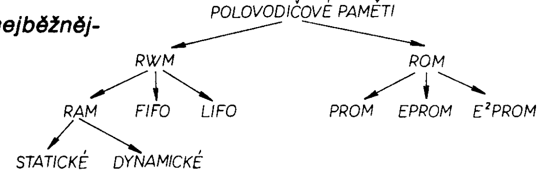
Tak jako základním prvkem sekvenčního obvodu je klopný obvod, můžeme za stavební kámen číslicové

techniky považovat paměťovou buňku. Obojí mají schopnost uchovat, zapamatovat si binární stav logické proměnné. Zatímco klopný obvod je víceméně „samostatným“ prvkem, tvoří paměťová buňka jen část číslicové paměti. Teprve to je konstrukční prvek, schopný uchovat určité množství logických hodnot nebo informací pro další potřebu nebo zpracování. Využití paměti je podstatou variability a modifikovatelnosti jednoduchých i složitých elektronických číslicových systémů. Funkci dílčích obvodů i celých systémů lze ovlivňovat (programovat) změnou obsahu užitých paměťových prvků. Principem funkce takto programovaných automatů i nadále zůstává přesně definovaná sekvence obvodových stavů. Není však již určena výlučně zapojením, ale v podstatné míře obsahem „programové“ paměti.

Podle aplikačního určení a užité technologie se paměti dělí do několika skupin. My se zaměříme na oblast unipolárních polovodičových pamětí, které zcela dominují. Popíšeme si nejdůležitější a nejzajímavější typy, protože tuzemská literatura je v tomto ohledu značně skoupá.

Typy polovodičových pamětí

Základní rozdělení typů polovodičových pamětí podle aplikace je na obr. 29. Paměti lze rozdělit do dvou hlavních skupin:



Obr. 29. Symbolické dělení nejběžnějších typů pamětí

paměti RWM (Read Write Memory) jsou určeny jak pro zápis, tak čtení uložených dat;

paměti ROM (Read Only Memory) slouží pouze pro čtení pevně uložených dat. Užívají se jako paměti programu a konstant.

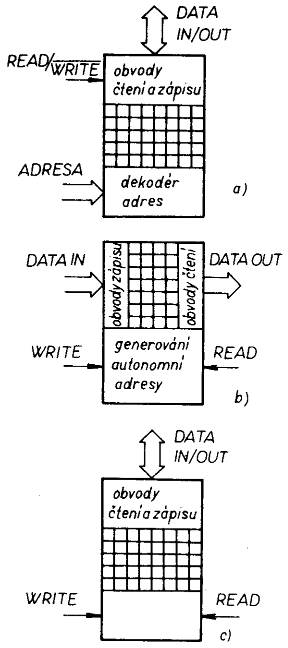
Oba tyto základní typy lze pak podle dalších kritérií dělit do dalších, specializovaných oblastí.

Prvním takovým kritériem je schopnost uchování paměťového obsahu po odpojení napájecího napětí. Volatilní paměti v tom případě svůj obsah nenávratně ztrácejí, jsou to zpravidla všechny paměti typu RWM, nevolatilní paměti, tvořené pamětmi typu ROM, si svůj obsah uchovávají.

Paměti typu RWM se podle přístupu k datům dělí na:

a) Paměti RAM (Random Acces Memory — s libovolným přístupem). Data mohou být do paměti zapisována a z paměti čtena v libovolném pořadí. Každá konkrétní datová položka je vždy určena příslušnou adresou, která musí datový přístup vždy doprovázet. Prakticky všechny samostatně vyráběné paměti RWM jsou typu RAM, obr. 30a.

b) Paměti FIFO (First In — First Out). Data mohou být z paměti čtena pouze v tom pořadí, v jakém byla do paměti zapsána. Systém FIFO nevyžaduje díky tomu externí adresování, obr. 30b.



Obr. 30. Základní typy pamětí RWN; a) RAM, b) FIFO, c) LIFO

Každá zapsaná položka může být čtena pouze jednou, pak je ztracena. Na tento typ pamětí si vzpomeňte později v souvislosti s vytvářením datových a adresových front.

c) Paměti LIFO (Last In — First Out). Pořadí čtení uložených dat je přesně opačné, než v předchozím případě, obr. 30c. Této paměťové konstrukce se opět využívá jako vratného zásobníku ve složitějších monolitických strukturách.

Jako poslední se užívá všeobecně známé dělení pamětí RAM na statické a dynamické. Obdobně se paměti ROM dělí na skutečné ROM a EPROM. Kromě toho ještě existují některé méně známé, přesto však užitečné a vyráběné paměti, z nichž pozornost zasluhují zvláště typy IRAM, NVRAM a EEPROM.

Stručný přehled základních unipolárních technologií

K dosažení velké hustoty integrace (LSI), přímo vázané s potřebou co nejmenšího příkonu, jsou obvody exponovaných bloků mikropočítačů (jako mikroprocesory, paměti různého typu nebo periferní a specializované složité obvody) v současné době realizovány některou z unipolárních technologií MOS (Metal Oxid Semiconductor).

V podstatě existují tři základní technologie MOS — PMOS, NMOS a CMOS, obr. 31. Jejich označení přímo vyplývá z typu kanálu tranzistoru MOS, který se tou kterou technologií vytváří.

Technologie PMOS, obr. 31a, vytváří p-kanálové tranzistory difúzí p-dopantů (akceptorů — obvykle bóru) do křemíkového substrátu typu n. Všechny kanály (tedy i drain a source) jsou vytvořeny současně v jedné vrstvě.

Technologie NMOS, obr. 31b, je obdobná, k vytvoření n-kanálových tranzistorů se však užívá n-dopantů (donorů — fosfor, arzén), difundovaných do substrátu typu p.

CMOS, tj. komplementární technologie MOS kombinuje obojí, p i n-kanálové prvky na společném křemíkovém substrátu, obr. 31c. Pro vytvoření komplementárních tranzistorů musí být do původního nejprve selektivně nadifundován opačný typ substrátu. Teprve pak je možno začít s realizací jednotlivých tranzistorových kanálů.

Většina z prvních obvodů MOS a paměťových prvků byla realizována technologií PMOS. S rostoucími požadavky na rychlost a hustotu integrace se stále více uplatňovala technologie NMOS. V současné době nacházejí nejširšího uplatnění různé varianty technologie CMOS, jejíž největší předností je jednotkový příkon. Technologická zlepšení (HMOS, CHMOS) již prakticky odstranila původní nedostatek, kterým byla menší rychlost vůči NMOS. I když by podrobnější rozbor technologií byl jistě zajímavý, věnujeme se raději obdobnému rozboru struktury unipolárních pamětí. Jistě ne každý má postačující představu o tom, jaké typy těchto pamětí v současné době existují a jaké jsou jejich principy.

Základním požadavkem na jakýkoli paměťový prvek v tuhé (pevné) fázi je co nejmenší příkon na hradlo a co největší dosažitelná hustota integrace na jedné, co největší přístupová rychlost na druhé straně. Tomu nejvíce vyhovují technologie CMOS.

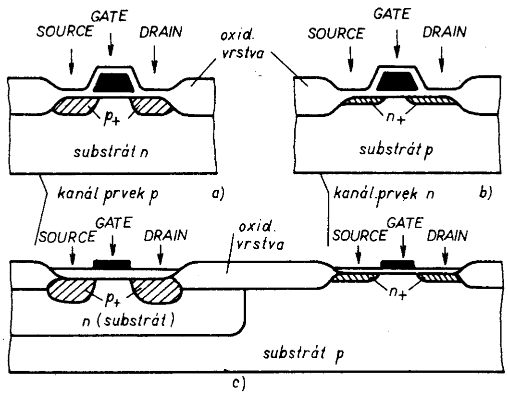
Paměti EPROM

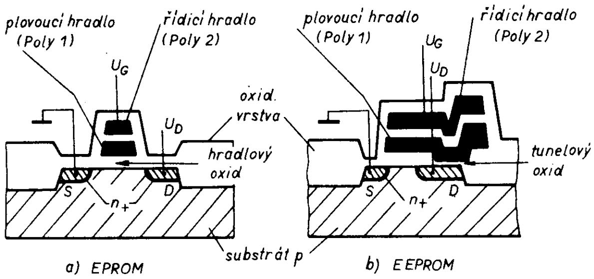
Erasable and Programmable Read Only Memory je typ paměti nevolatilního typu (její obsah zůstává zachován i bez přítomnosti napájecího napětí). Programují se elektrickými impulsy, jejich obsah může být vymazán ultrafialovým zářením pouze jako celek. Vyrábějí se i bez mazacího „okénka“. V tomto provedení mohou být naprogramovány pouze jednou.

Typická vnitřní struktura buňky EPROM je na obr. 32a. Základní kanál se vytváří běžným způsobem. Potom se dvojúrovňovým procesem vytvářejí dvě polykrystalická křemíková hradla, na nichž je založen vlastní nevolatilní paměťový princip. Horní hradlo slouží k výběru buňky v řádku paměti. Vnitřní hradlo je plovoucí (floating), nemá žádný vývod. Obě hradla jsou jak vůči sobě, tak vůči kanálu dokonale izolována vrstvami oxidů.

Paměť EPROM se programuje vstřikováním „horkých“ elektronů z kanálu, pronikajících při zvětšeném (programovacím) napětí kanálu do izolovaného, plovoucího hradla. Tím plovoucí hradlo získá náboj, určující stav příslušné buňky. Náboj plovoucího hradla je kapacitně vázán na potenciál hradla vstupního. Ten se mění při výběru buňky podle toho, je-li pasívní, nebo právě adresovaná. Součet těchto dvou potenciálů určuje stav jednotranzistorové buňky aktivovaného kanálu. Je-li plovoucí hradlo dostatečně nabito, posouvá se prahové napětí tranzistoru tak, že se buňka (bit) jeví naprogramovaná do žádoucího stavu (L). V opačném případě zůstává prahové napětí malé, buňka si zachovává původní stav (H), shodný se stavem po vymazání jejího obsahu.

Ultrafialové záření při mazání paměti indukuje do plovoucího hradla dosta-





Obr. 32. Schéma technologie polysilikonové struktury paměťových buněk EPROM a EEPROM

Obr. 31. Princip tří základních unipolárních technologií; a) PMOS, b) NMOS, c) CMOS

tek energie k tomu, aby byla překročena energetická bariéra izolačního prostředí kolem něj. Elektrony, tvořící náboj hradla, se proto mohou rozptýlit do vstupního hradla a kanálu buňky.

Stručně shrnuto, programování i mazání obsahu buňky EPROM se uskutečňuje nabíjením a vybíjením plovoucího hradla.

Paměti EPROM musí být samozřejmě před programováním nejprve vymazány. Paměti lze mazat i programovat až po vyjmutí paměti ze zařízení.

Paměti EPROM se standardně vyrábějí s paralelním datovým portem 8 bitů. Vnitřní strukturu těchto pamětí si popíšeme na příkladu dnes už vlastně nejjednodušší „epromky" 2716, obr. 33. Její paměťové buňky jsou uspořádány do typické matice, v tomto případě 128 řádků × 16 sloupců. Každému sloupci přísluší 8 paměťových buněk, aktivovaných současně jako byte. Přístup k matici je řízen adresovými dekodéry. Nižší část adresy se dekóduje na sloupce, vyšší část na řádky matice. Řádkovým dekodérem jsou ovládána řídicí hradla jednotlivých osmic paměťových buněk, jejich příslušné kanálové oblasti jsou pak řízeny spoluprací výstupů sloupcových dekodérů a čtecích nebo programovacích obvodů.

Aktivita těchto pamětí EPROM se ovládá kombinací dvou řídicích signálů, chip enable (CE) a output enable (OE). V módu čtení obsahu paměti se úrovní CE = L uvádí obvod do aktivního režimu, úrovní OE = L se pak aktivují 3stavové výstupy. Při úrovni CE = H je paměť uvedena do tzv. stand-by (pohotovostního, vedlejšího) módu, ve kterém je pasívní, její příkon se však zmenšuje asi na čtvrtinu jmenovitého odběru.

V módu zápisu do paměti se nejprve na špičku PRG přivádí napětí asi +25 V, na vstup OE úroveň H. Ta umožňuje přivést na 8bitový datový port programovací data. Zápis na každé adresované místo paměti se realizuje přivedením impulsu TTL L→H→L→ o trvání přibližně 50 ms na vývod CE. Odpovídající specifikace již zajišťuje programátor pamětí automaticky, detailní údaje k jednotlivým typům pamětí je nutno hledat v katalogu.

Přístupová doba běžných pamětí EPROM je asi 200 až 450 ns, kapacita jednoho paměťového obvodu je od 2 kB do 64 kB, je tedy poměrně značná a blíží se kapacitě dynamických pamětí RAM. Hlavní výhody, možnosti opakovaného programování obsahu, se využívá s výhodou nejen na vývojových pracovištích, ale i jako náhrady trvalých pamětí PROM v menších a středních výrobních sériích.

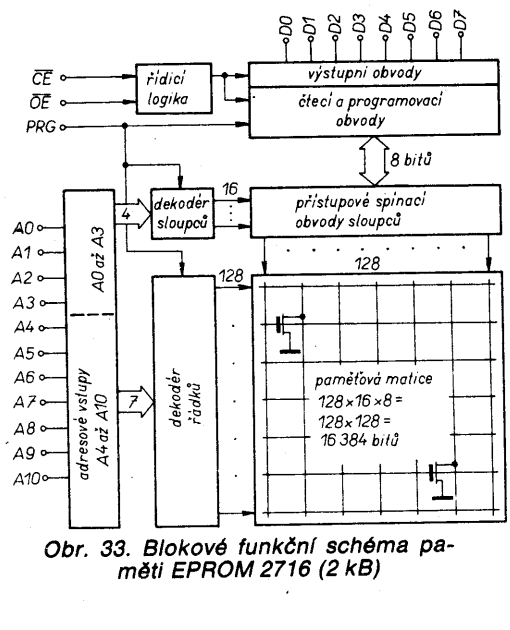
Paměti EEPROM

Princip mazatelných i programovatelných pamětí E^2PROM je do značné míry podobný předchozímu. Podstatný rozdíl ukazuje srovnání příčného řezu paměťovou buňkou (obr. 32b), zvláště v oblasti plovoucího hradla. Zatímco oblast kanálu je od hradla vzdálena zhruba 50 nm, v oblasti elektrody drain se tato vzdálenost zmenšuje na nepatrnou vrstvičku „tunelového oxidu", asi 10 nm. Vzájemná vzdálenost obou hradel je již opět větší, kolem 80 nm.

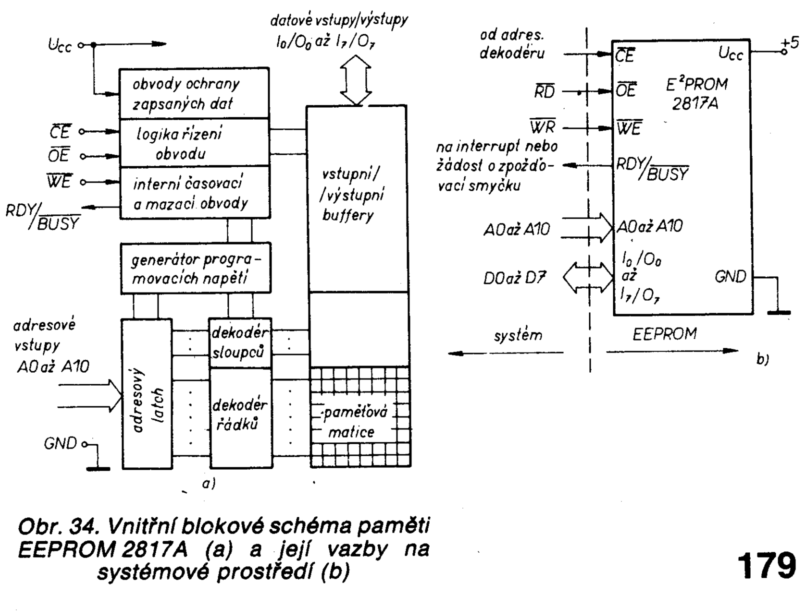
Podstata programování paměti E^2 PROM je opět založena na ovládání prahového napětí unipolárního tranzistoru změnou náboje jeho plovoucího hradla. V tomto případě je však hradlo elektrickou cestou nejen nabíjeno, ale také vybíjeno. Hlavní roli při tom hraje právě ona tenká vrstvička oxidu, která umožňuje na principu Fowler-Nordheimova fenoménu nábojovým nosičům (elektronům) pronikat touto izolační vrstvičkou při dosažení energie zakázaného pásu, která je podstatně menší než energie, potřebná k překročení energetické bariéry izolačního prostředí standardním mechanismem.

Uvedený princip umožňuje, aby paměti E^2PROM byly elektricky programovány i mazány přímo v zařízeních, v nichž jsou používány (In-Circuit). Je samozřejmé, že nemusí být mazán celý obsah současně, ale je možno postupovat po jednotlivých bytech.

I když informace ze zahraniční literatury dávají tušit, že vývoj těchto pamětí není ukončen, někteří výrobci je vyrábějí již poměrně dlouho. Co nás nedávno překvapilo, je paměť 2817A a kapacitou 2 kB, s jediným napájecím napětím 5 V, kterou vyrábí fa Intel; její vnitřní struktura je na obr. 34. Z hlediska softwarové i hardwarové implementace je zcela nenáročná, protože obsahuje jak interfaceovou logiku, tak pomocné zdroje programovacích/mazacích potenciálů (řešené na principu nábojové pumpy) a navíc všechny potřebné časovací obvody. Průběh vnitřní činnosti se tak stává autonomní a nijak nezatěžuje procesor zařízení. Vnitřní stav je pouze signalizován signálem RDY/BUSY. Standardní signály řízení paměťového přístupu CE, OE jsou doplněny nezbytným signálem



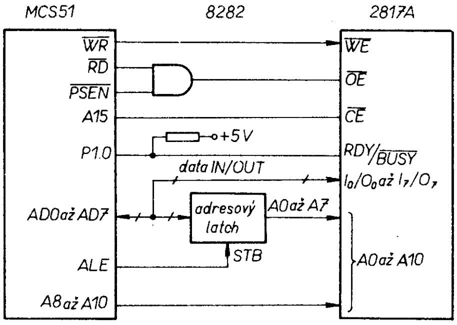
Obr. 33. Blokové funkční schéma paměti EPROM 2716 (2 kB)



Obr. 34. Vnitřní blokové schéma paměti EEPROM 2817A (a) a její vazby na systémové prostředí (b)

$\overline{WE}$, rozlišujícím podle úrovně mezi zápisem (L) a čtením (H) adresovaného byte paměti. Důležitou součástí paměti je obvod pro ochranu uložených nebo zapisovaných dat, který zastavuje přístup do paměti a činnost vnitřních obvodů při zmenšení napájecího napětí pod určitou mez.

I když se nemůžeme zabývat detaily, nemohli jsme si odpustit ukázat, jak jednoduchá je implementace této paměti do struktury mikropočítače. Na obr. 35 je její použití jako nevolatilní programové paměti s měnitelným obsahem (programem) pro některou variantu jednočipového mikropočítače z řady MCS-51. Latch 8282 je užit v obvyklé funkci demultiplexeru adres A0 až A7. Odvození řídicího signálu $\overline{OE}$ jako logického součinu $\overline{RD}$, $\overline{PSEN}$ umožňuje spolu se signálem $\overline{WR}$ = $\overline{WE}$ ovládat čtení i zápis do paměti. Po nastavení řídicích signálů $\overline{CE}$, $\overline{WE}$ = L, $\overline{OE}$ = H, což znamená příkaz k zápisu, 2817A automaticky ukládá adresy, data i řídicí signály a začíná se zápisem. Po dobu jeho trvání je datový bus zcela uvolněn, což umožňuje procesru provádět jiné úlohy. Tento interval je indikován signálem RDY/$\overline{BUSY}$ = L.



Obr. 35. Implementace obvodu 2817A jako programové nevolatilní paměti s měnitelným obsahem v sestavě souboru MCS-51

Nevolatilní charakter pamětí E^2 PROM, vylučující potřebu zálohování, je v řadě případů neocenitelnou předností. Byly již realizovány vzorky pamětí s kapacitou 256 kB.

Paměti SRAM

Standardní paměti tohoto typu mají paměťové buňky buď ze čtyřtranzistorových (výhodné z hlediska ceny), nebo šestitranzistorových (výhodné z hlediska minimalizace příkonu) jednotek, obr. 36. V obou uspořádáních vždy jedna dvojice tranzistorů zajišťuje vazbu maticové buňky na řádkový a sloupcový dekodér.

Předností statických pamětí RAM je to, že jejich obsah nemusí být oživován. Přístupová doba může být velmi krátká, asi 50 až 200 ns. Poměrná složitost buňky však omezuje možnost dosáhnout velkých paměťových kapacit na čipu.

Do nedávné doby se rychlé paměti SRAM vyráběly technologiemi NMOS, popř. HMOS. Jejich nedostatkem byl poměrně velký příkon a malá paměťová kapacita. Ty paměťové bloky, které bylo třeba zálohovat, se osazovaly obvody CMOS. Jejich kapacita však byla malá, desky s pamětmi CMOS často obsahovaly několik desítek IO.

Současnost přináší v oblasti statických pamětí CMOS zvrat. Díky novým technologiím se dosahuje dříve nepředstavitelných kapacit čipu. Např. obvody HM6116 a HM6264 fy Hitachi, která patří k pionýrům v této oblasti, obsahují paměti SRAM s kapacitou 2 kB, popř. 8 kB v jednom pouzdru. Jejich přístupové doby se, podle specifikace, pohybují v rozsahu 120 až 200 ns, příkon v režimu stand-by je menší než 10 µW (ve specifikaci LP) a mimořádně usnadňuje zálohování volatilního paměťového obsahu.

Hitachi navíc tyto paměti SRAM vyrábí jako rozměrově, vývodově i kapacitou zcela shodné se standardními pamětmi EPROM 2716 a 2764. Po dodatečném ošetření jediného řídicího signálu R/$\overline{W}$ je pak možno programovou paměť EPROM nahradit pamětí SRAM. To umožňuje podstatně pružnější oživování a odlaďování mikropočítačového systému.

Paměti SRAM jsou standardním prvkem číslicové techniky a jako takové se vyrábějí v nejrůznějších provedeních, s různou kapacitou, s oddělenými nebo multiplexovanými vstupy/výstupy a s různou šířkou datového pole.

Různým koncepcím pamětí SRAM odpovídají i určité odchylky v organizaci řídicích signálů. Ty jsou však vždy snadno pochopitelné.

Paměti DRAM

Ačkoli se kapacita pamětí SRAM díky novým technologiím stále zvětšuje, zůstává v důsledku rozsáhlé konstrukce paměťové buňky SRAM stále o jednu generaci za dosažitelnou kapacitou dynamických pamětí DRAM, vyráběných stejnou technologií. Ty naopak nemohou dosáhnout přístupové rychlosti statických pamětí. Paměti DRAM byly až do současné doby jediným efektivním prostředkem k dosažení větší kapacity operační paměti. Nástup rozsáhlejších pamětí CMOS RAM a samozřejmě i řada dalších, systémových činitelů bude i do budoucna znamenat trvalý tlak na další zvětšování paměťové kapacity čipů.

Dynamická paměť RAM využívá jako paměťového média, podobně jako paměť EPROM, elektrického náboje. Ten však v tomto případě nemá trvalý charakter a musí být, pokud možno v pravidelných, minimálních časových intervalech, cyklicky obnovován. To je jak technologicky, tak obvodově náročný problém. Ve srovnání s předchozími se u pamětí DRAM setkáváme s neobvyklou realizací paměťové buňky, složitým přístupem k zápisu a čtení jejího obsahu i jeho udržování a nakonec se zcela odlišným adresováním.

Paměti DRAM se vyrábějí jako jednobitové. To znamená, že pro výstavbu datové paměti 1 byte je zapotřebí 8 pouzder IO. Paměti různých kapacit jsou vyráběny se vzájemně kompatibilními vývody, liší se pouze konkrétním využitím adresového pole, které by však při klasické interpretaci, obvyklé u pamětí RAM, bylo velmi rozsáhlé. Např. paměť 4164 s kapacitou 2^{16} = 64 kb vyžaduje 16bitovou adresu. Pro omezení počtu vývodů (tím i zmenšení rozměrů pouzdra IO, ceny, zástavné plochy na desce s plošnými spoji . . .) se u pamětí DRAM využívá adresového multiplexu. Adresa je do adresového latche, obr. 37, zapisována nadvakrát. Nižší část adresy, odpovídající řádku paměťové matice, se zapisuje sestupnou hranou strobovacího impulsu $\overline{RAS}$ (Row Address Strobe), vyšší část, odpovídající sloupcům, stejnou hranou impulsu $\overline{CAS}$ (Column Address Strobe). Tak lze paměť DRAM až do kapacity 256 kb umístit do 16vývodového pouzdra DIL. Celá paměťová matice, rozložená po ploše čipu, nebývá realizována jednolitě, ale rozkládá se do několika shodných polí: např. na obr. 37 vidíme dvě paměťová pole, každé má 128 řádků a 64 sloupců. Obě pole jsou při paměťovém přístupu adresována současně, aktivní výběr (selekt) mezi oběma poli z hlediska datového přístupu zajišťuje nejvyšší bit (A6) adresového latche.

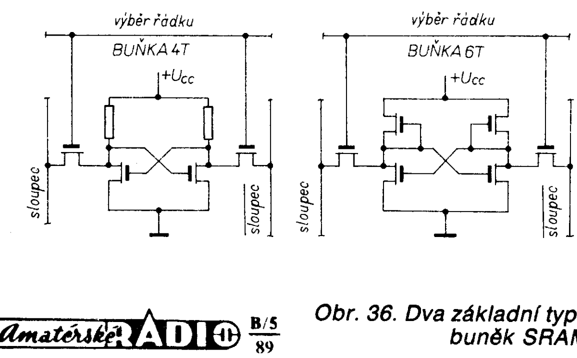
V podstatě existují tři základní paměťové cykly — read (čtení), write (zápis) a $\overline{RAS}$ only refresh (oživení).

Rozlišení mezi cykly read a write umožňuje řídicí signál $\overline{WE}$. Pro zápis musí být $\overline{WE}$ = L.

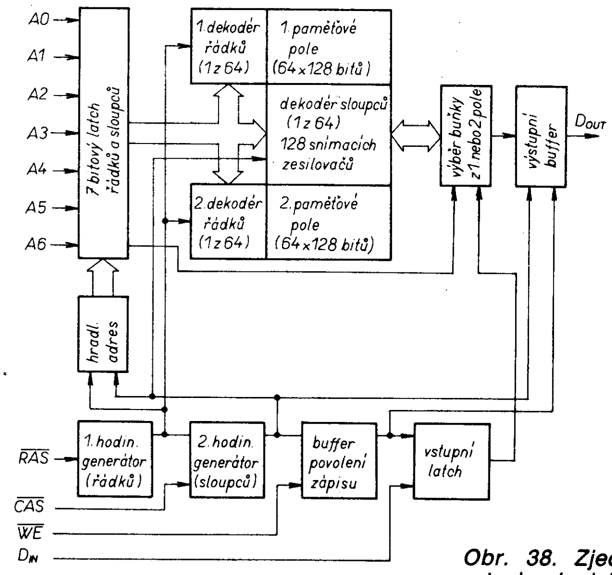
Víme již, že základní funkcí signálu $\overline{CAS}$ je vzorkovat ukládané sloupcové adresy. Jeho druhou funkcí je to, že působí jako enable pro datový výstup za předpokladu, že je aktivní i signál $\overline{RAS}$. V opačném případě je datový výstup ve třetím stavu. Tento mechanismus umožňuje sdílet společný datový bit několika pamětmi DRAM a tak vytvářet rozsáhlejší paměťové pole.

Při zápisu se data, přivedená na vstup D_{IN}, ukládají do paměti sestupnou hranou buď signálu $\overline{CAS}$ nebo $\overline{WE}$, podle toho, která z nich je poslední. Když hrana $\overline{WE}$ předchází $\overline{CAS}$, což je obvyklý případ, označovaný jako „early write", zapisuje sestupná hrana $\overline{CAS}$. Opačná zápisová posloupnost se nazývá „late write". Zápis „early write" umožňuje přímo propojit špičky D_{IN}, D_{OUT} a využít je jako společného třístavového vstupu/výstupu. Přístup typu „late write" v některých případech umožňuje dosáhnout rychlejšího zápisu, datové špičky D_{IN}, D_{OUT} však musí být elektricky izolovány.

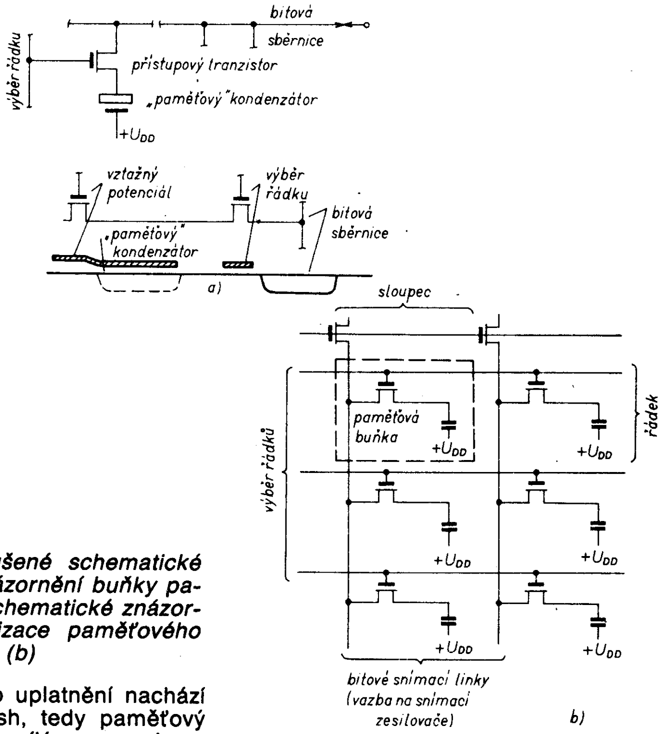
Zvláštní pozornost zasluhuje problém oživování paměti. Zde bude vhodné věnovat trochu pozornosti vnitřní struktuře paměťové buňky (obr. 38a). To, že bylo dosaženo velké hustoty integrace, umožnila jednoduchost buňky. Buňka se skládá z vázané dvojice řízeného vstupního tranzistoru



Obr. 36. Dva základní typy paměťových buněk SRAM



Obr. 37. Blokové znázornění vnitřní struktury dynamické paměti RAM 16k×1 bit, 4116



Obr. 38. Zjednodušené schematické a technologické znázornění buňky paměti DRAM (a) a schematické znázornění vnitřní organizace paměťového pole (b)

a paměťové kapacity. Hodnota bitu, uloženého v buňce, je prezentována přítomností nebo absencí elektrického náboje. Paměťová kapacita je selektivně zpřístupňována impulsním řízením vstupního, „přístupového" tranzistoru. Náboj může být na paměťovém „kondenzátoru" udržen vzhledem ke svodům a migraci pouze omezenou dobu. Při užívání pamětí DRAM proto musí existovat mechanismus, který periodicky s určitým intervalem (řádově ms) čte obsah paměťové buňky a protože samo čtení je destruktivní proces, obnovuje podle zjištěného stavu úplným nabitím nebo vybitím paměťového kondenzátoru původní logickou úroveň buňky. Obvod, který čte, vyhodnotí a obnoví původní náboj buňky se označuje jako snímací zesilovač.

Nyní si již můžeme osvětlit vnitřní organizaci paměťového pole (obr. 38b). Každý sloupec buněk je přes izolační obvod ovládán ze sloupcového dekodéru prostřednictvím bitové snímací linky (BSL). Tato linka je vždy opatřena jedním snímacím zesilovačem. Protože se při libovolné adrese aktivuje jeden řádek a jeden sloupec, může být čtena nebo zapisována pouze jediná paměťová buňka. Protože však je každý sloupec vybaven vlastním snímacím zesilovačem, oživují se současně všechny buňky na adresovaném řádku. Aby nastalo zotavení (refresh) celé matice, musí být zpřístupněny (oživeny) všechny její řádky ve zmíněném časovém intervalu. Musí být postupně aktivovány všechny kombinace adresových vstupů — u paměti na obr. 37 je to $2^7 = 128$ kombinací.

Z hořejšího odstavce vyplývá jedna z okrajových možností oživování paměti DRAM využitím periodického cyklu read, write nebo read-write-modify. Jednu z mimořádně vtipných ukázek včetně využití stránkového módu nabízí řešení paměti videoram Spectra. Bohužel, v běžných případech podobným přístupem bezpečné zotavení všech paměťových řádků během několika ms zajistit nelze. Navíc je vždy nutno zabránit střetu zotavení (refresh) s požadavkem datového přístupu, popř. je nutno tuto situaci řešit.

Podstatně širšího uplatnění nachází RAS — only refresh, tedy paměťový cyklus, v němž se vysílá pouze adresa řádku a tento řádek je oživen. Protože není aktivován signál CAS, nedochází v cyklu ani ke čtení, ani k zápisu. Této metody, spolu s uplatněním logického arbitru možných přístupových kolizí, užívají speciální řadiče dynamických pamětí, např. 8203 fy Intel.

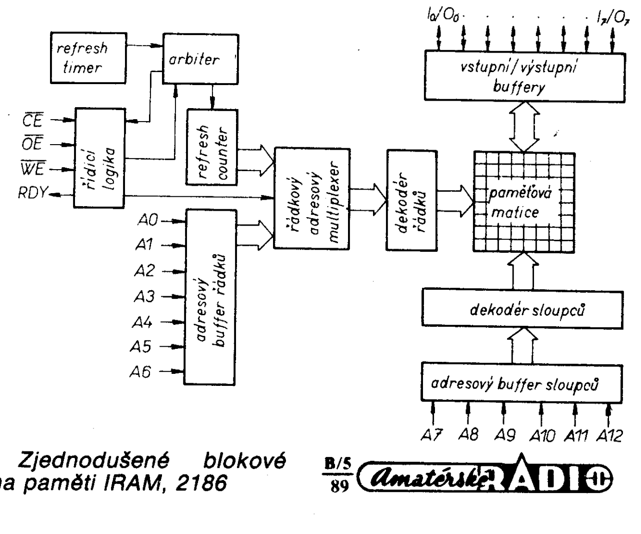
Zajímavě a pro jednodušší aplikace postačujícím způsobem řeší zotavování paměti DRAM mikroprocesor Z80. Je vybaven generátorem refresh adres, které vysílá v tzv. synchronním taktu RAS — hidden refresh jako platnou adresu oživování na nižších adresových výstupech A0 až A6. Tato adresa je vysílána skrytým způsobem ve volné době cyklu FETCH, kdy je volná adresová sběrnice a mikroprocesor dekóduje čtenou instrukci. Platnost vysílané adresy refresh je strobována signálem RFSH na špičce 28 Z80. Užitý princip je ukázkou, jak obejít potřeby externího přístupového arbitru.

Paměti IRAM

V poslední době se množí náznaky aktivity předních výrobců v oblasti komplexního řešení monolitických kombinací pamětí DRAM s velkou kapacitou včetně úplných obvodů řadiče. Bohužel, konkrétní údaje zatím publikovány nejsou. Snad mohou určitou představu, vhodnou také pro dokreslení problému oživování pamětí DRAM, poskytnout integrované paměti IRAM, které jsou vlastně něčím podobným, ale v „menším" měřítku. Vyráběla je již dříve firma Intel, ale zdá se, že bez většího úspěchu.

Na obr. 39 je blokové schéma paměti IRAM 2186 s kapacitou 8 kB. Dynamická RAM, tentokrát vzhledem k relativně malé kapacitě paměti adresovaná přímo, bez multiplexování, je doplněna časovačem (timer) on chip, adresovým refresh čítačem, přístupovým arbitrem a jediným výstupním signálem Ready. Arbiter řeší všechny problémy, vznikající při současném externím přístupu do paměti a požadavku interního cyklu refresh. Vždy, když refresh naruší žádost o přístup k paměti, přechází kontrolní výstup open Ready→L. Jeho prostřednictvím vkládá procesor stavy WAIT.

S těmito atributy se paměti IRAM chovají jako statické paměti RAM. Při omezených kapacitách jim ovšem vyrostla těžko zdolatelná konkurence v pamětech CMOS RAM. Otázkou zůstává, jaký bude v oblasti pamětí IRAM další vývoj. Zdá se, že takové obvody, možná již řešené jako specializované, by v budoucnu mohly integrací paměťového bloku včetně úprav adresování, refreshe a všech pomocných



Obr. 39. Zjednodušené blokové schéma paměti IRAM, 2186

funkcí přinést další radikální zjednodušení paměťových obvodů mikropočítačů.

Paměti NVRAM

Každá buňka nevolatilní paměti RAM se vlastně skládá vždy ze dvou buněk — jedné SRAM, druhé E^2PROM. Odtud vyplývá, že z hlediska dosažitelné paměťové hustoty na tom paměti NVRAM nebudou nejlépe. NVRAM se v běžné činnosti chová jako statická paměť RAM, zajišťuje však uložení dat při odpojení napájecího napětí. Odvozený interní signál, hlídající zmenšení nebo odpojení napájecího napětí, zajišťuje přepis obsahu buněk RAM do jim odpovídajících buněk E^2PROM.

Paměť NVRAM může spolupracovat s mikroprocesorem jako běžná statická paměť RAM, její obsah je však nevolatilní, nezávislý na napájecím napětí.

Vytváření paměťových sestav

Z jednotlivých paměťových obvodů IO se k dosažení větší kapacity, případně i datové šíře, vytvářejí rozsáhlejší paměťové bloky RAM, ROM buď na společných, nebo samostatných deskách s plošnými spoji. Přitom je vždy nutno zajistit dokonalé adresování právě jediného obvodu, který má být aktivován. Jinak by vznikaly vzájemné kolize dvou nebo několika datových vstupů/výstupů. V předchozích odstavcích jsme viděli, že většina paměťových obvodů zachovává určitou standardní úpravu všech vstupních a výstupních signálů, které lze rozdělit do tří skupin, datové, adresové a řídicí.

Pro vzájemnou vazbu paměťového bloku s komunikujícím systémem se využívá systému sběrnic (busů). Jsou to vždy několikabitové komunikační cesty, jejichž funkce, signálové parametry a vzájemná součinnost odpovídají určitým konvencím.

Adresová sběrnice je jednosměrná, u malých systémů maximálně 16bitová. Nejvyšší adresové bity se vyhrazují pro výběr (selekt) jednotlivých paměťových IO. K tomu se využívá rychlých bipolárních adresových dekodérů.

Datová sběrnice musí být vždy obousměrná, tedy třístavová. Nejčastěji se setkáváme se sběrnicemi 8bitovými.

Řídicí sběrnice umožňuje řídit komunikaci po datové sběrnici. Jejím prostřednictvím se řídí přístupy k jednotlivým blokům paměti, jejich časování, aktivace a potřebné systémové korespondence.

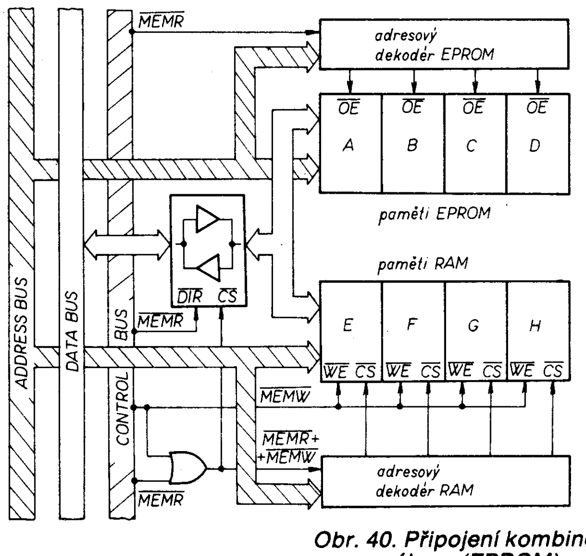
Se systémovou sběrnicí se blíže seznámíme později. V této fázi jí využijeme pouze pro jednoduchou ukázku vazby kombinovaného paměťového modulu RAM/EPROM na vnější prostředí (obr. 40).

K výběru pamětí RAM i EPROM se v obou sekcích využívá samostatných adresových dekodérů. Při naznačeném uspořádání musí každý dekodér rozlišit čtyři adresové bloky. Pro celkem osm odlišných bloků stačí dekódovat tři nejvyšší, pro další adresování nevyužité adresové bity, např. A13 až A15. Takové jednoduché řešení ovšem pevně definuje spojitý adresový prostor. Každému bloku IO přísluší úsek 8 kB, který při menší paměťové kapacitě RAM nebo EPROM nemusí být využit. Obě sekce RAM i EPROM mohou podle způsobu využití obou dekodérů buď spolu sousedit, nebo se i prolínat. U složitějších systémů, u nichž je třeba s paměťovým prostorem šetřit (nebo se musí počítat s potřebou jeho úpravy), musí být i řešení adresových dekodérů složitější.

Jednotlivé obvody EPROM jsou při vyslání příslušné adresy a žádosti o čtení MEMR vybírány dekodérem D-EPROM. Podobně aktivitu dekodéru D-RAM určuje vedle příslušné adresy buď žádost o čtení (MEMR) nebo zápis (MEMW) do paměti RAM. Zbývající, podstatná část bitů adresové sběrnice slouží k vlastnímu adresování paměťové matice obvodů RAM i EPROM.

Rozsáhlá třístavová datová sběrnice obou paměťových bloků RAM i EPROM je na obr. 40 řešena jako interní. Od systémové datové sběrnice (data bus) je oddělena bipolárním třístavovým obousměrným budičem. Řešení je výhodné jak z hlediska potřebné minimalizace zatížení výstupů obou stran sběrnice, tak také z hlediska hledání případných závad. K aktivaci obousměrného budiče sběrnice je opět třeba využít logického součtu přístupových žádostí k paměti (MEMR + + MEMW), k řízení směru přenosu musí být využito signálu MEMR.

Organizace přístupu k oběma paměťovým sekcím RAM i EPROM včetně přidělování adresových prostorů a odpovídajícího sekvenčního řízení vzájemné součinnosti všech tří částí systémové sběrnice je již záležitostí nadřazeného, například mikroprocesorového systému.

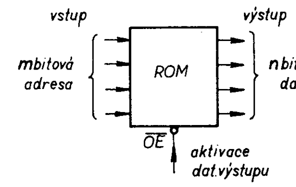


Obr. 40. Připojení kombinovaného programového (EPROM) a datového (RAM) paměťového bloku na systémovou sběrnici (příklad)

Aplikace paměťových obvodů v hardwarových automatech

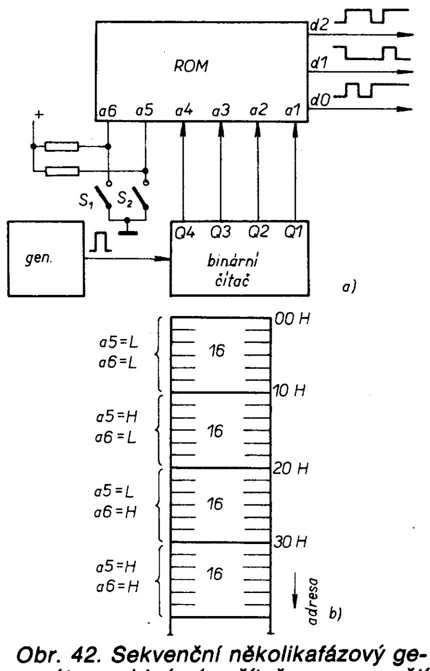
Uvažujme nyní, jak by se dalo využít vlastností paměťových obvodů při konstrukci běžných logických a sekvenčních automatů, založených na již dříve probraných principech.

Jako první příklad můžeme znovu uvažovat automaty s výlučně sekvenčním pracovním cyklem, tedy bez reakce na vstupní proměnné. Úlohy jsme se již dotkli při návrhu generátorů (obr. 18, 20), u nichž jsme pro návrh kombinační logiky tvorby budicích a výstupních proměnných využívali kombinované metody stavového grafu + pravdivostní tabulky. Zde celkem přirozeně vystupuje možnost nahradit tyto obvody pevně naprogramovanou nevolatilní pamětí (ROM, EPROM), použitou jako konverzní datový obvod (obr. 41). Adresa paměti je pak tvořena kódem vstupních, obsah příslušného paměťového místa kódem výstupních



Obr. 41. Náhrada kombinační logiky pamětí ROM

proměnných. Takové metody jsou v praxi poměrně časté, např. při realizaci nestandardních datových konverzí a při mapování adresových prostorů, v nichž nacházejí výhodné použití rych-



Obr. 42. Sekvenční několikafázový generátor s binárním čítačem a pamětí ROM; a) blokové schéma, b) znázornění možnosti výběru jednotlivých stránek ROM nastavením adresových bitů a5, a6

lé binární paměti. Ty ovšem mají zpravidla malou paměťovou kapacitu.

Na obr. 42 je možné řešení generátoru několikafázového impulsního signálu s neměnným pracovním cyklem, rozfázovaným pomocí libovolného, např. binárního čítače. Čítač, adresující v taktu hodinového signálu paměť ROM, představuje generátor vnitřních stavů. Možnost libovolně naprogramovat obsah paměti odstraňuje potřebu generování budicích funkcí. Pak ani není třeba řešit úlohu pomocí stavového grafu. Každý vnitřní stav čítače představuje aktuální adresu paměti ROM a obsah takto adresovaného paměťového místa vždy určuje příslušnou kombinaci, kód výstupních proměnných.

Z řešení vyplývá řada výhodných vlastností, například:

— délku pracovního cyklu automatu lze definovat, popřípadě ovládat nastavením pracovního cyklu čítače,

— okamžité stavy výstupních proměnných i jejich vzájemné relace v jednotlivých pracovních fázích lze libovolně měnit, bez úprav zapojení, přeprogramováním obsahu paměti,

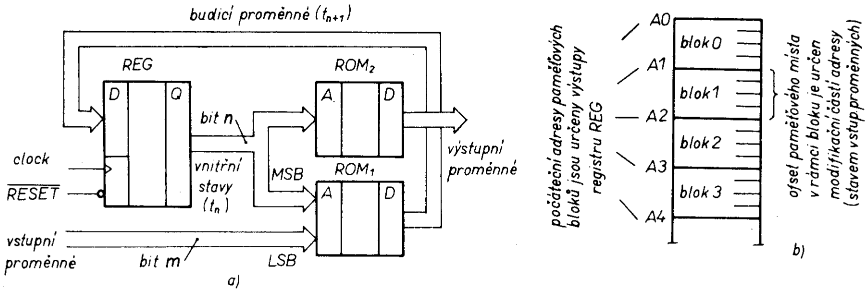
— celý proces je řízen v taktu hodinového signálu. Jeho každý impuls inkrementuje (zvyšuje o +1) obsah čítače a tedy i adresu paměti. Při použití obousměrného čítače, popř. čítače s přednastavením by bylo možno dosáhnout inverzní posloupnosti generovaných výstupních proměnných (dekrementováním stavu čítače), popř. nesekvenčních skoků stavů čítače a tím i sledu stavů výstupních proměnných. Zvláště druhá možnost zasluhuje zamyšlení, protože představuje jeden ze základních principů technické realizace programovatelných automatů,

— disponuje-li užitá paměť větším počtem adresových vstupů, než vyžaduje pracovní cyklus čítače, je možné vybírat z několika formátů výstupních funkcí. Na obr. 42b je znázorněna možnost statického výběru čtyř programů, ovládaného pomocí vhodného nastavení dvou nejvyšších adresových vstupů paměti — každá ze čtyř „stránek" pak může být naprogramována pro jinou výstupní funkci,

— rovněž je možné zastavit pracovní cyklus v libovolném stavu (wait) a opět jej spustit synchronním ovládáním přístupů hodinových impulsů na vstup čítače. Obecně je možné dosáhnout i volby mezi cyklickým a jednorázovým přechodem mezi jednotlivými stavy automatu atd.

Existují i další možnosti využití specifických rysů, které do obvodového řešení automatů zavádí aplikace paměťových prvků. S těmi, které jsme si uvedli, se však již budeme dále setkávat jako s charakteristickými obvodovými prvky a funkcemi programovatelných automatů a počítačů. Zvlášť velkou podobnost poznáme především mezi dvojicí stavový čítač — konverzní paměť ROM na obr. 42 a programový čítač — programová paměť libovolného programovatelného systému.

Základní uspořádání automatu na obr. 42 má jedno typické omezení. Sled výstupních akcí (kombinací výstupních proměnných) obvodu je prostřednictvím konverzní paměti ROM jednoznačně vázán na sled vnitřních stavů řídicího čítače. Pro jiné než sekvenční řazení vnějších akcí automatu musí být modifikována činnost čítače. To ovšem vyžaduje řadu dalších obvodů, které musí být především schopny nějakým



Obr. 43. Blokové schéma univerzálního logického automatu, využívajícího pamětí ROM (a), odpovídající využití paměťového prostoru (b)

vhodným způsobem vyhodnotit požadavek nesekvenční změny stavu čítače a zajistit její korektní provedení.

Postupme nyní dále a pokusme se využít pamětí ROM v obecném hardwarovém automatu, reagujícím na vstupní proměnné. Za východisko zvolíme koncepci, odpovídající obr. 15. Paměťovými prvky budeme nahrazovat obě kombinační sekce obvodů tvorby budicích a výstupních proměnných. Zásadním rozdílem proti předchozímu příkladu nyní bude to, že jako „řadič vnitřních stavů" nebude použit čítač. Nyní již nestačí generovat prostou sekvenci sousedních vnitřních stavů. Příslušný řídicí obvod musí být schopen, v závislosti na stavu vstupních proměnných, řídit skokové přechody na vnitřní stavy (tvořící základní složku adresy paměťového místa), odpovídající požadovaným výstupním akcím automatu.

Blokové schéma na obr. 43, vyhovující úloze, využívá dvou pamětí ROM. Zapojení lze funkčně rozdělit do dvou částí. První, představující základ obvodu, je tvořena *n*bitovým paralelním dynamickým registrem, zachycujícím po dobu t_{clock} vnitřní stav (t_n) a programovou pamětí ROM1. Adresový vstup této paměti je rozdělen do dvou sekcí. Hlavní, *n*bitová, je určena okamžitým stavem, t_n, odpovídá tedy výstupu registru. Druhá je určena stavem vstupních proměnných, které musí být se systémem synchronizovány. Využívá nejnižších adresových bitů. Datový výstup každého takto upraveného paměťového místa (lokace) ovládá zpětně vstup registru, čímž určuje budoucí proměnné následujícího stavu, t_{n+1}.

Nižší adresové bity, ovládané stavem vstupních proměnných, působí v daném uspořádání jako adresové modifikátory. Paměťový prostor ROM1 se tím rozděluje do bloků, v nichž je každému vnitřnímu stavu vyhrazen pevný počet paměťových lokací, rovný $x = 2^k$, kde k je počet vstupních proměnných. Např. pro dvě vstupní proměnné se každý blok skládá ze čtyř paměťových míst.

Nyní již bude popis funkce obvodu z obr. 43 jednoduchý. Předpokládejme, že obsah registru REG byl vynulován vnějším signálem RESET. Jeho výstupy tedy adresují nultý paměťový blok ROM1. V závislosti na stavu vstupních proměnných se upřesňuje adresa v rámci tohoto bloku. Na datové vstupy registru je zaveden obsah odpovídajícího paměťového místa, který určuje adresu následujícího paměťového bloku stavu (t_{n+1}). Každé paměťové místo v rámci paměťového bloku může být naprogramováno tak, aby v příštím stavu (t_{n+1}) nastal buď sekvenční posuv do sousedního, setrvání v původním nebo nesekvenční skok do libovolného paměťového bloku. Poloha skutečně adresovaného paměťového místa v rámci bloku je určena stavem vstupních proměnných. Druhá paměť, ROM2, pouze nahrazuje kombinační logiku pro konverzi vnitřních stavů obvodu na výstupní proměnné.

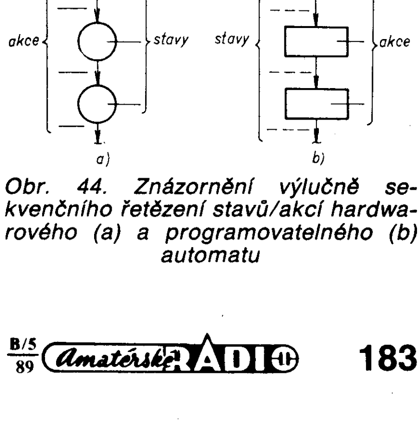
Tato koncepce (ve srovnání s předchozí) představující podstatně univerzálnější přístup k řešení automatu se nám opět později vybaví, až budeme uvažovat o funkci bloku instrukčního registru a dekodéru programovatelného automatu.

Programovatelný automat

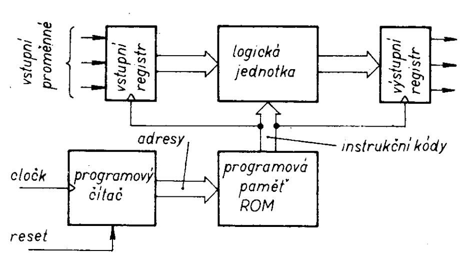
Oba do značné míry typické přístupy k řešení jednoduchého logického, popř. sekvenčního automatu v předchozích příkladech, založené na využití paměťových prvků, umožňují určitou, i když velmi omezenou funkční variabilitu obvodu změnou naprogramování obsahu pamětí. Jak jsme již naznačili a uvidíme dále, obě koncepce se uplatňují i v obvodovém řešení programovatelných automatů a počítačů, i když odlišným způsobem. Samy o sobě jsou sice logické, nemohou však znamenat podstatný přínos z hlediska univerzálnosti využití. Celý systém zůstává příliš primitivní, protože musí stále respektovat současnou stavovou vazbu všech vnějších a vnitřních proměnných.

Obecná koncepce programovatelných automatů je založena na zcela odlišné a lze říci přímo opačné filozofii. Pokusme se ji logicky odvodit tak, že ještě jednou budeme uvažovat o nedostatcích a omezeních, vyplývajících z možností hardwarových řešení automatů.

V souladu s obr. 44a uvažujme čistě sekvenční činnost hardwarového automatu, který z každého stavu t_n přechází do stavu t_{n+1}. Při přechodu mezi stavy musí vždy nastat nějaká, alespoň vnitřní akce, činnost. Pokud je významná i z hlediska vnějších obvodů, můžeme ji označit jako vnější akci. Sekvenční činnost hardwarového automatu není



Obr. 44. Znázornění výlučně sekvenčního řetězení stavů/akcí hardwarového (a) a programovatelného (b) automatu



Obr. 45. Jednoduchý sekvenční programovatelný automat

závislá na vstupních proměnných, provádí tedy pouze vnitřní a výstupní akce. Rozhodujícím kritériem, určujícím průběh těchto akcí, jsou tedy jednotlivé sekvence vnitřních stavů.

Na obr. 44b je obdobným způsobem stylizována sekvenční činnost programovatelného automatu. Je obdobná, automat přechází z jednoho do druhého vnitřního stavu provedením nějaké vnitřní či vnější akce. Zásadní rozdíl je však v tom, že programovatelný automat může nezávisle na stavu, v jakém se nachází, vykonat jakoukoli akci, v případě potřeby i akci vstupní. Realizace každé akce je určena instrukcí, tvořící základní prvek programu, uloženého v programové paměti. Po skončené každé akci dostává řídicí jednotka automatu novou instrukci, určující, jakou akcí bude ve své činnosti pokračovat. Význam vnitřního stavu automatu je tímto mechanismem potlačen. I když to neplatí zcela obecně, stává se pouhým předělem mezi dvěma operacemi, akcemi.

Činnost programovatelného automatu již není limitována požadavkem trvalého sledování stavu všech vstupních proměnných. Může probíhat v sekvenčních úsecích, z nichž se každý může zabývat určitými úlohami. Při jejich provádění pak sleduje pouze ty vstupní proměnné, které jsou pro aktuální činnost významné. Navíc může tyto proměnné buď v přímé, nebo upravené formě ukládat a dále zpracovávat jak pro aktuální, tak pozdější potřebu. Program, který řídí činnost automatu, může být založen na nesrovnatelně efektnějších a logičtějších postupech a algoritmech, než jaké umožňovaly technické prostředky hardwarového automatu.

To jsme se však již příliš „odvázali". Zkusme si popsat nejjednodušší programovatelný automat, který by navazoval na naše dosavadní představy. Nástin možného řešení je na obr. 45. Automat je pro jednoduchost koncipován tak, že umožňuje pouze sekvenční činnost. Skládá se v podstatě ze čtyř funkčních bloků:

— programového čítače,
— programové paměti,
— programovatelné logické jednotky,
— vstupního a výstupního registru.

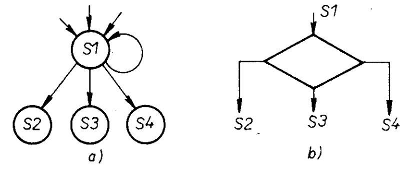
Programový čítač, řízený hodinovým signálem, adresuje programovou paměť. Program je tvořen sledem instrukcí. Instrukci si můžeme v tomto případě definovat jako binárně kódovaný obsah jednoho paměťového místa. Odlišnými instrukčními kódy lze ovládat jednak činnost logické jednotky, jednak čtení ze vstupního nebo zápis do výstupního registru vstupních a výstupních proměnných. Logickou jednotku si můžeme představit např. jako rozsáhlejší logický kombinační obvod, jehož funkce je modifikovatelná na základě aktuálního instrukčního kódu, přivedeného na řídicí vstupy jednotky. Naprogramováním paměti ROM lze v každém stavu programového čítače dosáhnout, v rámci instrukčního souboru, kterým disponuje logická jednotka, výběru libovolné logické funkce mezi vstupními a výstupními proměnnými.

Činnost tohoto primitivního automatu již v zásadě není určena jeho obvodovou strukturou, ale programem, uloženým v programové paměti. Univerzální programovatelný automat ovšem musí umožňovat nejen prostou sekvenční činnost, ale i jiná, nesekvenční řazení a provádění instrukcí. K tomu je samozřejmě zapotřebí podstatně složitějších technických i programových prostředků. Struktura i instrukční soubor programovatelného automatu musí umožňovat programové skoky, cykly, testy, práci s podprogramy, přerušení programu a řadu jiných funkcí.

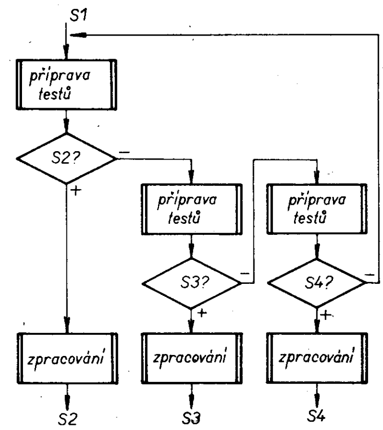
Tím vzniká zcela nová situace. Pro realizaci celé rozsáhlé třídy úloh může být užito prakticky jednotné struktury programovatelného automatu s definovaným instrukčním souborem. Funkce automatu v konkrétních aplikacích je podstatnou měrou určena programem. Byť je sebesložitější, je vždy realizován omezenou, konečnou množinou instrukcí, kterou disponuje užitý procesor. V té se dá orientovat mnohem snáze a systematičtěji, než ve struktuře hardwarového automatu, vyvíjené prakticky stále znovu od A do Z. Celá práce je také podstatně „přívětivější". Lze dále využívat vyšších programových prostředků, jazyků, vývojových systémů i prostředků pro ladění programů, které výrobci jednotlivých automatů, včetně rozsáhlé literatury nabízejí. Při systematické práci s konkrétními typy procesorů pak lze mnoho úloh řešit rutinní cestou, s využíváním postupně získávaných zkušeností.

Logickým důsledkem těchto možností a předností je vznik mikroprocesoru (μP). Je to integrovaný procesor programovatelného automatu, kterému by na obr. 45 odpovídala kombinace programový čítač (vzdáleně představující primitivní sekvenční „řadič") a logická jednotka (zhruba nahrazující „operační jednotku").

Podle aplikace a stupně technického vývoje samozřejmě existují různé typy programovatelných automatů. Dva základní si dále podrobněji popíšeme.



Obr. 46. Nesekvenční větvení sledů stavů i akcí hardwarového automatu (a), programový přepínač řízený na základě soustavy testů příznaků předchozích akcí (b)

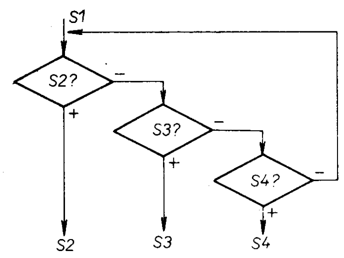


Obr. 47. realizace přepínače vloženými akcemi a testy, které umožňuje jednoduchý programový řadič

Vzájemné rozdíly v řešení jejich procesorů osvětlují následující obrázky. Na obr. 46 se ještě naposledy vracíme k hardwarovému automatu. Je na něm znázorněno větvení sledu vnitřních stavů. Ze stavu S1 může automat na základě kódu vstupních proměnných přejít do jednoho ze stavů S2 až S4, nebo setrvat ve stavu původním. Celý tento, v hardwarovém pojetí složitý blok, v podstatě tvoří programový přepínač, znázorněný symbolicky na obr. 46b. Podle toho, jaký stav vstupních proměnných je vyhodnocen, vybírá se cesta, vedoucí k provedení požadované akce.

Na obr. 47 je orientačně znázorněn možný postup realizace takového přepínače programovatelným automatem, který užívá tzv. jednoduchý obvodový řadič. Tento typ μP je schopen v každém taktu hodinového signálu zpracovat jednu instrukci, která je však schopna zajistit pouze elementární operaci. Tou se myslí například nastavení přenosové cesty adresy, čtení, zápis, provedení logického testu apod. Provedení akce pak zpravidla vyžaduje sekvenci většího počtu těchto elementárních instrukcí. V takové situaci je, před zahájením programového úseku, simulujícího přepínač, nejprve nutno pomocí těchto instrukcí uskutečnit celou řadu operací, na jejichž základě mohou být zahájeny testy jednotlivých větvení. Po testu na platnost podmínky se buď vystupuje z cyklu na přípravu provedení požadované akce, nebo se přechází do dalšího testu. Není-li splněna žádná z podmínek, uzavírá se smyčka zpět na vstup celého programovaného úseku a tam probíhá tak dlouho, dokud není některá z podmínek splněna. Nedostatek obvodového řadiče pro běžné aplikace je evidentní. Je jím potřeba značného počtu instrukcí pro uskutečnění úplné akce. To značnou měrou komplikuje návrh programu, protože programátor se musí soustavně zabývat množinou nadbytečných „mikroinstrukcí".

Zmíněné problémy radikálním způsobem potlačují běžné procesory, vybavené tzv. mikroprogramovaným řadičem. Instrukce těchto jednotek jsou tvořeny sekvencí, posloupností „mikroinstrukcí" ve smyslu předchozího odstavce. Jsou proto podstatně výkonnější a umožňují přehlednější výstavbu programu. Řešení programového úseku přepínače se pak, při využití testů příznakových indikátorů, může zjedno-

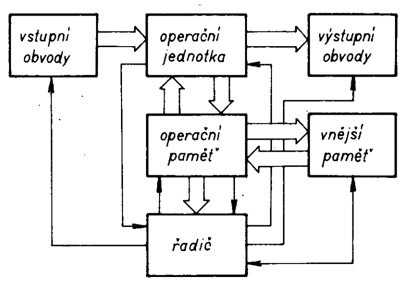


Obr. 48. Mikroprogramovaný řadič potlačuje nutnost programovat redundantní (nadbytečné) akce

dušit až do formy, naznačené na obr. 48. Nelze však zapomenout na to, že provedení komplexní instrukce zpravidla trvá podstatně déle.

Přednosti programovatelných automatů mohou být zastíněny prakticky pouze ve dvou případech. Prvním jsou mimořádně jednoduché aplikace. Druhý případ se stává aktuálním tehdy, když se jedná o prioritu extrémně rychlé časové odezvy. Z principu činnosti programovatelného automatu, založeného na postupném řešení úlohy, postupné realizace sledu instrukcí vyplývá, že hardwarový automat by měl být vždy rychlejší. Nebývá to však vždy pravdou, protože řada úloh je již natolik složitá, že je prostě přímou hardwarovou cestou řešit nelze.

Programovatelnost automatu, který může variabilně testovat vybrané proměnné, kritické podmínky a stavy, ukládat a zpracovávat různá data a datové bloky, modifikovat svoji činnost a nejrůznějším způsobem komunikovat s vnějším prostředím znamená, že jeho činnost může být mimořádně efektivní. Princip programového řízení ovšem zdaleka není novinkou posledních let. Řadu hlavních zásad navrhl již před 150 lety Babbage. Na konci 2. světové války pak základní principy koncipoval von Neumann. Princip jeho počítače je blokově znázorněn na obr. 49.



Obr. 49. Von Neumannova koncepce číslicového počítače

Celá činnost je podle příslušného programu řízena řadičem, který spolu s operační jednotkou tvoří základní prvky procesoru. Ústřední jednotkou celého procesoru pak je ALU (aritmeticko-logická jednotka — unit). Program, který má být realizován, je uložen v operační paměti. Tam je také vyhrazen prostor pro data, která jsou systému zadávána, nebo která si vytváří sám jako výsledek čtení a zpracování vstupních dat nebo interních operací. Paměťový prostor programu a dat (ROM, RAM) může být buď spojitý, nebo oddělený (Harwardská koncepce). Činnost celého počítače je synchronizována hodinovým signálem. U klasického von Neumannova systému se program realizuje postupným zpracováváním jednotlivých instrukcí, uložených podle konvence krokového adresování ve spojitém paměťovém prostoru bezprostředně jedna za druhou.

Každá instrukce se zpracovává v tzv. instrukčním cyklu. Tento typický proces lze rozdělit do několika charakteristických, vzájemně navazujících etap:

a) čtení instrukce,
každá aktualizovaná instrukce musí být nejprve přečtena z paměti a uložena do instrukčního registru IR;

b) provádění instrukce,
potom musí být dekódován operační kód instrukce a na jeho základě generována sekvence elementárních mikroinstrukcí, řídící všechny činnosti, potřebné ke korektnímu provedení instrukce. To se týká jak interních obvodů procesoru (operační jednotka), tak i vnějších bloků (paměti, obvody IO . . .), s nimiž procesor komunikuje prostřednictvím systémové sběrnice;

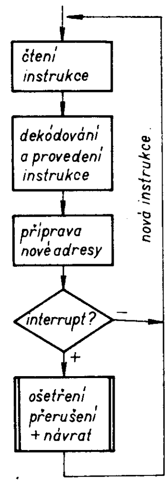
c) příprava adresy následující instrukce,
ještě v průběhu aktuálního instrukčního cyklu se již připravuje adresa instrukce následující. Von Neumannova koncepce při tom umožňuje generovat adresu bezprostředně následující, sekvenčně řetězené instrukce zcela automaticky a extrémně jednoduše, prostým inkrementováním stavu programového čítače, který dosud určoval adresu platné instrukce. Tím se podstatně zrychluje běh programu a zjednodušuje programování.

U instrukcí, porušujících sekvenční sled ovšem musí být z pravidla učiněna výjimka — skokové instrukce, volání podprogramů apod. Procesor pak musí být schopen adresovat požadované paměťové místo, u volání podprogramů nebo aktivace přerušení pak ještě navíc uložit návratovou adresu do tzv. zásobníkové paměti (stack);

d) test požadavku přerušení,
při aplikaci programovatelného automatu se prakticky vždy vyskytují takové speciální požadavky nebo vstupní proměnné, které se vymykají běžným zvyklostem. Především v tom, že vyžadují bezprostřední, naléhavé ošetření s minimální přípustnou časovou prodlevou, prakticky v reálném čase. Normální režim programovatelného automatu těmto požadavkům nevyhovuje, protože je také v podstatě vůbec nerespektuje. Naopak, běžné vstupní proměnné testuje pouze tehdy, když v průběhu provádění programu potřebuje znát jejich stav. Vzniklá časová prodleva pak ovšem může být nepřijatelná. Je sice možné např. periodicky testovat stav některé proměnné. Tím se však podstatně zmenšuje výkonnost, propustnost systému, protože kvůli testu se vždy pozastavují ostatní činnosti.

Pro daný účel bývá většinou procesor vybaven zvláštním vstupem, umožňujícím uplatnit tzv. externí přerušení (interrupt). Požadavek přerušení může být vůči systému zcela asynchronní, náhodný. Aby bylo dosaženo co nejrychlejší reakce, testuje se vždy na konci každého instrukčního cyklu, zda tento požadavek nebyl uplatněn. Pokud ne, pokračuje program další instrukcí. Pokud ano, je nejprve dokončena instrukce běžícího programu a ukládá se obsah programového čítače do zásobníkové paměti jako návratová adresa běžícího programu. Technickými prostředky (řadič přerušení) se vyhodnotí priorita žádosti a vyvolá se tzv. vektor přerušení. Je to jedna z několika adres, na které musí být předem uložen počátek programového ošetření příslušné žádosti. Jakmile je zpracování přerušení ukončeno, lze vyvoláním návratové instrukce přejít zpět k realizaci původního přerušeného programu.

Typický průběh instrukčního cyklu tak, jak byl popsán, postihuje vývojový diagram na obr. 50.



Obr. 50. Znázornění průběhu instrukčního cyklu

Vývojový diagram, metajazyk, strukturogram

Automat, který může být programován prostřednictvím instrukčního souboru, umožňuje systematický přístup k řešené úloze. Je k tomu vybaven potřebnými prostředky. Problémem, který nejen zůstává, ale naopak je danou situací zpravidla ještě zvýrazňován, je jeho vlastní naprogramování pro konkrétní úlohu. To není možno přímo řešit na úrovni jeho instrukčního souboru. Vlastní naprogramování mikropočítače je až konečnou, nejnižší úrovní celého postupu, což ovšem není nic nového.

Při řešení jakékoli úlohy musíme nejprve najít nebo znát vhodný postup (algoritmus), vedoucí k žádanému cíli. Může jich být víc. Pak volíme takový postup, který vzhledem ke konkrétní situaci považujeme za nejvhodnější. Víme, že ne vždy se rozhodneme správně. Z toho je vidět, že většinou musíme uvažovat několik alternativních řešení, která nejprve rozebíráme v hrubých rysech a teprve pak postupně, hierarchicky zpřesňujeme až na úroveň, při níž se rozhodneme o konečném postupu.

Jako názornou analogii takového hierarchického řešení můžeme uvažovat postup při řešení obyčejné slovní úlohy. I tu si musíme nejprve rozmyslet, dělat si vhodné poznámky. Teprve pak se snažíme o její matematický nebo logický zápis, který by již měl mít nějakou vhodnou strukturu. Na jeho základě sestavíme třeba rovnici, případně přímo dosadíme do známého vzorce a pak již úlohu řešíme podle obecného, vžitého a známého postupu, algoritmu. Řešení tedy probíhá v něko-

lika úrovních, vždy s využitím různých přístupů a prostředků. Ne každý bude úlohu řešit stejně. Záleží to na postupu, který zvolí, na konkrétních dostupných prostředcích, schopnostech a zkušenostech. Přesto, i když nelze řešení konkrétních úloh jednoznačně metodicky určit, je vždy možno využívat nebo přebírat zkušenosti, které řešitel sám nebo někdo jiný získal na třídě úloh daného typu.

Vždy, i když často ne uvědoměle nebo správně, postupujeme při řešení složitějších úloh metodou top-down, jejich postupným rozkladem shora dolů. Stejný musí být i postup při programování řešení úlohy na počítači. Úlohu nejprve definujeme v hrubých rysech, pak ji rozkládáme do programových bloků. Pak přecházíme postupně k jejich hierarchicky méně významným částem a nakonec k detailům. Vazby mezi jednotlivými bloky by měly být co nejjednodušší a odpovídat určitým pravidlům.

Určitým problémem při přenosu obdobného postupu do roviny programování je vhodná forma popisu postupně rozvíjené představy řešení, spojená s obtížným hledáním kompromisu mezi přehledností celku a přesným postihem detailů na konci řešení.

Na nejvyšší úrovni se problém popisuje vhodnou, často symbolickou formou, ale i slovně, např. v pomocném metajazyce, odvozeném vhodnou účelovou úpravou vyššího programovacího jazyka. Definují se datové a programové struktury. S postupným rozkladem hrubých rysů úlohy do hierarchicky méně významných bloků a nakonec detailů se velmi často uplatňují grafické formy znázornění. Představují již podklad pro zápis programu, ať již v některém vyšším jazyce nebo v asembleru (jazyce symbolických adres).

Teď jsme se však již dostali mimo rámec příspěvku. Jako vhodnou literaturu doporučujeme zejména knihu [5], která se uvedenou problematikou zabývá soustavně a přístupnou formou.

Vhodnými prostředky pro grafické znázornění vyvíjených programových bloků jsou tzv. strukturogramy a vývojové diagramy. Strukturogramy ovšem dosud nejsou příliš vžité, nabízejí však řadu výhodných vlastností, zejména pokud jde právě o možnosti zápisu a přehlednosti postupného strukturovaného rozkladu a „zjemňování" programových bloků. Jsou opět systematicky probírány v hezké knížce [6], věnované úvodu do jazyka BASIC. Nejvžitější formu symbolického znázornění programové posloupnosti, kterou jsme již na několika místech použili i my, představují vývojové diagramy. Ačkoli neposkytují přímou ochranu proti nestrukturovanému zápisu, jsou názorné a po získání určitých zkušeností se stávají dobrým nástrojem. Základní symboly a příkazové struktury, používané ve vývojových diagramech a strukturogramech, jsou na obr. 51.

Programovatelný logický automat

Pro první kontakt s koncepcí programovatelného logického automatu jsme vybrali řešení, založené na využití booleovského (CMOS) procesoru MC14500. Především pro jeho relativní jednoduchost, kterou vhodně navazuje na naše dosavadní úvahy. MC14500 ještě stále není mikroprocesorem ve vžitém slova smyslu, obsahuje však již jeho podstatné funkční bloky a rysy.

Podle obr. 52 si nejprve stručně popišme vlastní logický procesor. Má velmi jednoduchý, pouze 4bitový operační kód, umožňující definovat instrukční soubor s pouze $2^4 = 16$ instrukcemi (obr. 53). Nedisponuje aritmetickými instrukcemi. Kromě základních logických, přístupových a přesunových instrukcí v souboru nacházíme pouze instrukce skoku, přeskoku a návratu. Neobvyklá, pouze jednobitová datová sběrnice odpovídá výlučné aplikaci v logickém automatu s jednoduchou logickou jednotkou LU. Všechny logické a přesunové operace vždy mají jeden operand uložen ve výsledkovém registru, který je obdobou klasického akumulátoru ACC. Tam je ukládán i výsledek logické operace. Druhý z operandů je vždy přímý. Systém vůbec neužívá datovou paměť RAM, což je důsledkem orientace na jednobitové proměnné. Procesor také není vybaven možností změny adresové sekvence, příslušné funkce musí být v případě potřeby zajištěny externím obvodem.

Činnost automatu je synchronizována interním hodinovým generátorem procesoru. Jeho řídicí jednotka CU tvoří ve spolupráci s registrem instrukcí IR a dekodérem instrukcí ID systémový řadič.

Instrukce, přepsaná z externí paměti ROM s hranou hodinového impulsu do 4bitového registru IR, ovládá po dekódování v ID řídicí jednotku, která řídí následující činnosti:

— ovládá stav podmínkových klopných obvodů IEN, OEN a tím i přístupová hradla Read (umožňuje přístup přímého operandu na vstup LU) a Write (vznik zápisového impulsu do výstupní vyrovnávací paměti).

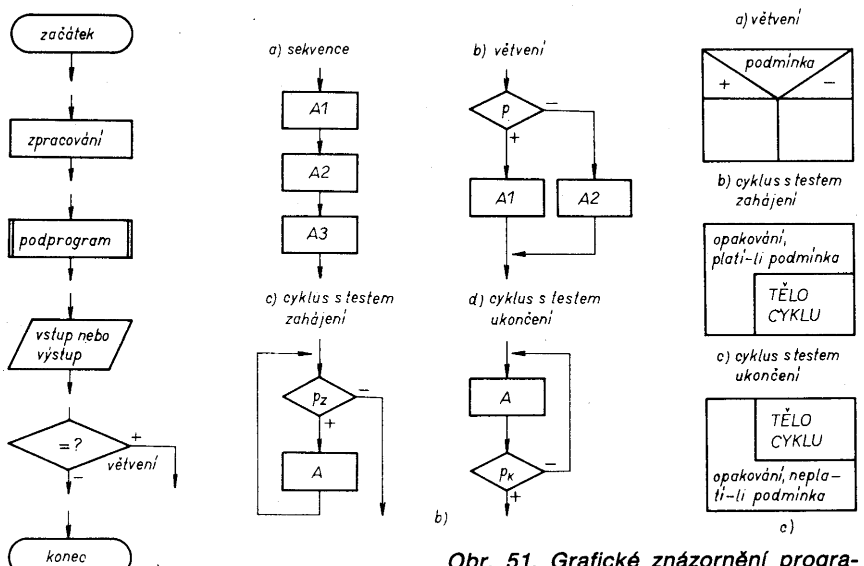
— řídí odpovídající funkci logické jednotky LU a multiplexeru MUX, umožňujícího volbu mezi přímým (Q) a inverzním ($\overline{Q}$) přístupem registru ACC na datovou sběrnici,

— usnadňuje řízení obvodu pro nastavování externího programového čítače PC při potřebě využití skokových instrukcí JMP, SKZ a RTN.

Pro realizaci programovatelného automatu je procesor MC14500 třeba doplnit několika externími obvody, jejichž konkrétním řešením lze určitým způsobem systém přizpůsobovat konkrétním požadavkům. Je to především paměť programu a s ní související programový čítač.

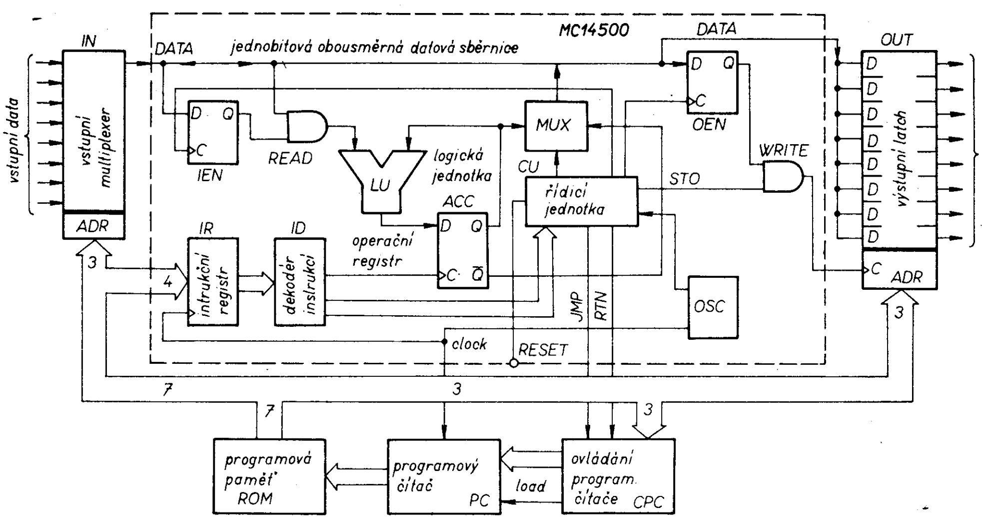
Při jednoduchém sekvenčním řetězení instrukcí je s každým taktem hodinového signálu inkrementován PC. Takto adresované paměťové místo přesouvá svůj obsah (4 bity) jako instrukci do procesoru IR a tam se interpretuje. Pokud paměť programu disponuje širším datovým polem, může být redundantních paměťových bitů využito k dalším účelům. Hlavní, všeobecné využití při tom nacházejí především pro adresování vstupního multiplexeru IN a výstupního latche OUT. Pro naznačené rozšíření jednobitové datové sběrnice procesoru na 8bitová vstupní/výstupní data postačují tři redundantní paměťové bity, jejichž vhodným kódováním v každé instrukci je možný výběr libovolné vstupní nebo výstupní proměnné.

Struktura logického procesoru neumožňuje (s výjimkou přeskokové instrukce), ale pouze podporuje možnost zavést skokové instrukce. Instrukce JMP a RET mohou být interpretovány různě podle toho, jak je řešen příslušný externí obvod ovládání stavu programového čítače CPC. Ten v případě těchto instrukcí musí zajistit nastavení PC na skokovou, cílovou adresu, případně ještě zajistit „úklid" běžné adresy jako adresy návratové. Možnosti řešení jsou různé. S určitou výhodou lze opět využít redundantních paměťových bitů jako adresových vektorů, umožňujících výběr z několika skokových adres. Při tom ovládání externích obvodů podporují příslušné řídicí signály, vyvedené z procesoru pro externí využití.



Obr. 51. Grafické znázornění programových bloků; a) základní symboly vývojového diagramu, b) vývojové diagramy sekvence, větvení a strukturovaných cyklů, c) strukturogramy větvení a cyklů

Těmito problémy se blíže zabývat nebudeme, všimneme si raději možností využití takto koncipovaného automatu. Omezený instrukční soubor sice radikálně zjednodušuje strukturu procesoru, ale zároveň omezuje pole jeho využití pouze na jednoduché aplikace. Protože neumožňuje aritmetické instrukce, je ponechán i problém řešení složitějších příkazových struktur na hardwarové inciativě aplikátora. I v jednoduchých aplikacích však pro-



Obr. 52. Zjednodušené blokové schéma programovatelného logického automatu s booleovským procesorem

Instrukce	Instr. kód	Popis instrukce	Operace
NOPO	0000	No change in registers	ACC → ACC, FLAGO → ⎍
LD	0001	Load result register	DATA → ACC
LDC	0010	Load complement	$\overline{DATA}$ → ACC
AND	0011	Logical AND	ACC DATA → ACC
ANDC	0100	Logical AND complement	ACC $\overline{DATA}$ → ACC
OR	0101	Logical OR	ACC + DATA → ACC
ORC	0110	Logical OR complement	ACC + $\overline{DATA}$ → ACC
EXNOR	0111	Exclusive NOR	if ACC = DATA, 1 → ACC
STO	1000	Store	ACC → DATA, WRITE → ⎍
STOC	1001	Store complement	$\overline{ACC}$ → DATA, WRITE → ⎍
IEN	1010	Input enable	DATA → IEN
OEN	1011	Outout enable	DATA → OEN
JMP	1100	Jump	JMP flag → ⎍
RTN	1101	Return	RTN flag → ⎍
SKZ	1110	Skip next instruction	if ACC = 0
NOPF	1111	No change in registers	ACC → ACC, FLAG F = ⎍

Obr. 53. Tabulka instrukcí logického procesoru MC14500

gramování takového automatu není příliš jednoduché, protože musí být brány v úvahu všechny elementární vnitřní akce. Uvažme např. realizaci logického součinu dvou vstupních proměnných. Postupně je nutno, při současném uvažování adresování vstupního multiplexeru, zavést první proměnnou na datovou sběrnici (IEN), pak ji přenést a zapsat do výsledkového registru (LD), znovu naadresovat a vyzvednout druhý operand (IEN), provést příslušnou logickou operaci (AND), pak výsledek přepsat na datovou sběrnici (STO) a teprve pak, při současném adresování redundantními bity zapsat jediný bit výsledku do výstupního latche (OEN). Takové programování je vlastně „mikroprogramování“, protože vlastní program se v záplavě pomocných instrukcí ztrácí. Koncepce MC14500 odpovídá struktuře již diskutovaného procesoru s jednoduchým obvodovým řadičem, provádí jednu instrukci v průběhu jednoho taktu hodinového signálu.

Na druhé straně nelze koncepci automatu s jednocyklovými operacemi jednoznačně zavrhnout. Pevný instrukční formát a relativně malý počet málo výkonných instrukcí, přístupová koncepce load/store a jednocyklové operace jsou atributy stále více diskutované koncepce RISC (Reduced Instruction Set Code). Nedostatek složitého programování v tomto případě odpadá, protože architektura RISC je založena výlučně na práci s kompilovaným kódem. V takovém případě se ukazuje, že optimálního překladu z vyššího jazyka do strojového kódu z hlediska rychlosti cílového programu lze dosáhnout tehdy, je-li instrukční soubor počítače na nejnižší úrovni. Pak je také jednoduchý průběh instrukčního (prakticky strojního) cyklu, a jednocyklové instrukce mohou být prováděny velmi rychle. S tím ovšem souvisí celá řada dalších problémů, jako jsou propustnost systémových sběrnic nebo doby paměťových přístupů, promítající se v obvodovém řešení (Memory Management Unit, Caché . . .).

Běžné mikroprocesory, s nimiž se v praxi setkáváme, využívají mikroprogramovaných řadičů. To by ovšem ke změně funkčních vlastností mikropočítače nestačilo. Další požadované vlastnosti můžeme odvodit nejlépe z toho, co se nám na řešení předchozího procesoru nelíbí.

Běžný mikroprocesor musí být především podstatně univerzálnější. Musí mít podstatně větší adresovatelný paměťový prostor, v němž musí být schopen nejen sekvenčního, ale i skokového pohybu. Musí být schopen dělat programové skoky a cykly, větvit program. To je možné pouze na základě testů podmínek. Musí mít tedy výkonnou aritmeticko/logickou jednotku, aby byl schopen dělat i matematické výpočty. K tomu je zapotřebí, aby si mohl „pamatovat“ vstupní proměnné, mezivýsledky i konečné výsledky pro další použití. Systémová paměť tedy nemůže být tvořena pouze pamětí programu (ROM, EPROM), ale i dat (RAM). K nim musí mít mikroprocesor odpovídající přístup. Obdobný přístup musí mít i k vnějšímu prostředí (vstupy, výstupy, přerušení). Existence systému přerušení je podmíněna existencí mechanismu zásobníkové paměti. Jestliže považujeme za samozřejmé, že mikroprocesor musí být vybaven dostatečně výkonným instrukčním souborem, zdálo by se, že k tomu, abychom si popsali, jak takový mikroprocesor a mikropočítač pracuje, jsme si již vytvořili všechny předpoklady. Myslíme si však, že by to byla chyba. Doposud jsme si neřekli nic o tom, jak mikroprocesor interpretuje číselné hodnoty a jak s nimi provádí základní aritmetické operace.

Číselné soustavy a kódy

Dosud jsme si mohli dovolit zcela přecházet problém definice číselné hodnoty a její interpretace v binární soustavě. V podstatě jsme se, až na drobné výjimky, zabývali pouze jednoduchou, elementární proměnnou, která mohla nabývat logických hodnot 0 nebo 1, tedy bitem.

Zaměřme se nejprve pouze na celá kladná čísla, tedy na absolutní hodnotu binárního čísla. Takové číslo lze vytvořit řetězenou skupinou bitů, v níž každému bitu přísluší podle pozice určitá váhová hodnota. Pokud má bit na této pozici hodnotu 1, je hodnota platná, při hodnotě 0 je nulová i váhová hodnota této pozice. Číselný rozsah, který může být interpretován, je přímo úměrný počtu bitů ve skupině.

Skupina osmi bitů představuje nejužívanější binární formát, byte (slabiku). Váhová hodnota pozice byte je v principu ekvivalentní mechanismu běžné dekadické soustavy, samozřejmě s tím rozdílem, že binární a dekadická soustava mají různé číselné základy (tj. 2 a 10).

V dekadické soustavě je poziční forma zápisu interpretací hodnoty součinu koeficinetu a váhy jednotlivých pozic. například

10^3	10^2	10^1	10^0
2	4	0	3

$$= 2 . 10^3 + 4 . 10^2 + 0 . 10^1 + 3 . 10^0.$$

Obdobné je i vyjádření hodnoty v binárním kódu. Koeficient každé pozice však může být vzhledem k základu soustavy pouze 0 nebo 1, váha nejnižšího bitu celého čísla je rovna 2^0. Například

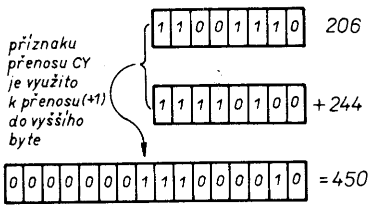
2^3	2^2	2^1	2^0
0	1	0	1

$= 0 . 2^3 + 1 . 2^2 + 0 . 2^1 + 1 . 2^0 = 5_{10}$.

Maximální hodnota, která může být interpretována jedním bytem je rovna

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	$=255_{10}$
128	64	32	16	8	4	2	1	

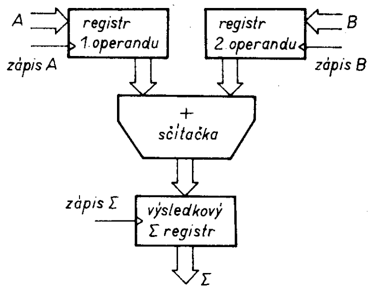
Čím širší bitový formát má skupina bitů, tím větší číselný rozsah (větší číselná hodnota) může být zobrazen. V praxi se téměř výlučně užívá násobku osmi bitů (výjimečně čtyř bitů). Skupina dvoubytů tvoří word (slovo), skupina čtyř byte double word (dvojslovo), obr. 54.



*Obr. 55. **Výsledek součtu dvou byte větší než 255 může být uložen ve dvoubytovém řetězci (word)***

Tento jev, který může být indikován při různých operacích, se označuje jako CY-carry (přenos). Může být v řadě případů využit pro efektivní realizaci některých algoritmů: Nejjednodušší příklad je na obr. 55. Použijeme-li pro vyjádření výsledku součtu 2bytové pole, pak stačí příznak CY zapsat jako hodnotu nejnižšího bitu (pozice) ve vyšším bytu výsledkového registru.

Sčítání binárních čísel umožňuje úplná paralelní sčítačka s přenosem. Je to poměrně složitý kombinační obvod.

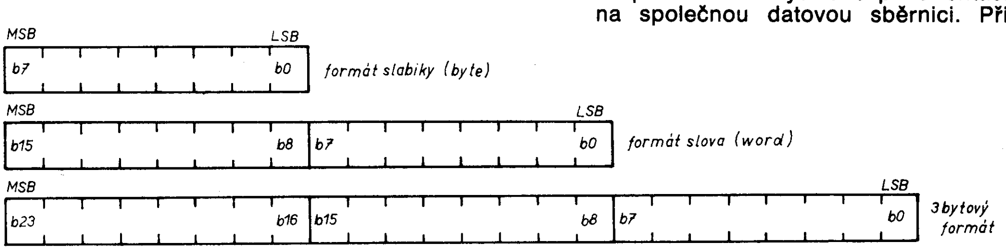


*Obr. 57. **Základní znázornění součtového obvodu***

Podstatou je to, že pro jeden operand a výsledek se používá společný registr, akumulátor. S ním jsme se již předběžně seznámili, viz obr. 52. Akumulátor (střadač) je ve skutečnosti tvořen dvojicí registrů, vlastním střadačem ACC a dočasným střadačem ACT. Pro druhý operand je k dispozici registr TMP.

Výhoda druhého uspořádání spočívá kromě jiného v jednoduchém přístupu k operandům a výsledku při orientaci na společnou datovou sběrnici. Při

*Obr. 54. Rozšiřováním informačního formátu (byte, word, 3 byte, **dword) se zvětšuje rozsah zobrazitelných hodnot***



Sčítání kladných čísel

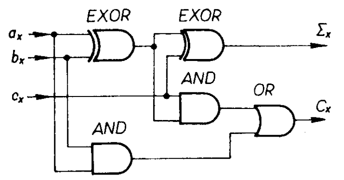
Aritmetické operace v binární soustavě jsou v podstatě založeny na stejném principu, jako operace ve vžité soustavě dekadické. Tak je tomu i se sčítáním. Rozdíl je ten, že hodnota každé pozice v dekadické soustavě se může pohybovat v rozsahu (0 až 9).10^x, kdežto v binární je pouze buď 0 nebo 1.2^x. V obou soustavách může být při sčítání nutný přenos z jedné pozice do druhé, vyšší. V dekadické je to tehdy, když koeficient příslušné pozice překročí mezní hodnotu, tj. 9. Pak se přebytek přičítá do vyšší pozice jako přenos.

Obdobně i v binární soustavě se používá přenos do vyššího řádu tehdy, když hodnota pozice „přestoupí" hodnotu 1. Příklad:

	2^3	2^2	2^1	2^0	
		1	0	1	
		0	1	1	5
	1	1	1	1 přenos	+3
CY=1		0	0	0	8

Platí pravidla:
0 + 0 = 0,
0 + 1 = 1,
1 + 1 = 0 + přenos,
0 + 1 + přenos = 0 + přenos,
1 + 1 + přenos = 1 + přenos.

Z příkladu rovněž vidíme, že při sčítání dvou 3bitových čísel byl vlivem přenosu z nejvyššího řádu překročen číselný rozsah — jde o tzv. přetečení.



vstup			výstup	
a_x	b_x	c_x	Σ_x	C_x
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

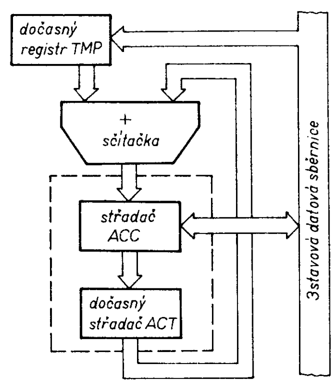
*Obr. 56. **Jeden stupeň paralelní binární sčítačky a jeho pravdivostní tabulka***

Příklad možné realizace pro jednu bitovou pozici je na obr. 56, funkci popisuje pravdivostní tabulka. Každá pozice sčítačky má tři vstupy — dva pro příslušné bity obou operandů (a_x, b_x) a jeden (c_x) pro přenos z nižšího řádu (x-1). Číselný výstup Σ_x slouží k zobrazení výsledku operace, výstup C_x pak představuje přenos do vyššího řádu (x +1), nebo, na nejvyšší pozici, do indikátoru přetečení CY.

Sčítačka je vysloveně kombinační obvod. Aby mohla pracovat, musí mít na oba datové vstupy přivedeny ustálené operandy. Také výsledek musí být někam uložen. Všechny tři funkce mohou být zajištěny registry, řízenými jednotlivými dílčími „mikrooperacemi" aritmetického součtu. Příklad obvodového uspořádání na obr. 57 jistě nevyžaduje komentáře. Je však možné i jiné zapojení, které má některé, na první pohled ne přímo zřejmé přednosti.

sčítání je třeba přesunout jeden operand do TMP, druhý do registru ACC; Přesun do ACC není nutný, je-li příslušný operand již uložen v akumulátoru jako výsledek předchozí operace. To je častý případ. Pak stačí pouze přepsat obsah ACC→ACT (jednoduchý interní zápis, bez potřeby datové sběrnice). Obsah ACT nyní tvoří druhý operand součtu a výsledek může být zapsán do ACC.

Obvodové řešení na obr. 58 je již základem standardního řešení střadačově orientované jednotky ALU univerzálního mikropočítače. Jeho jádro tvoří paralelní sčítačka s přenosem. V různých modifikacích naznačeného základního zapojení, s využitím řízení



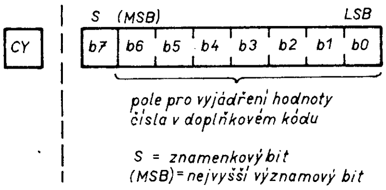
*Obr. 58. **Součtový obvod, využívající střadače a orientace na společnou datovou sběrnici***

nebo statického nastavení vstupů, může být sčítačka použita i pro jiné, např. logické a „posuvné" funkce.

Odčítání, doplňkový kód

Mohli bychom se nyní zabývat ukázkou binárního odčítání, ale neuděláme to, protože tak nepracuje ani aritmetická jednotka ALU. Vzhledem k technické realizaci (výhodné univerzální využití sčítačky) se využívá úpravy rozdílu na součet dvou čísel, z nichž jedno (menšitel) je záporné, např. 6 — 6 = 6 + (— 6). To však není možné při naší dosavadní definici binárního čísla, které chybí znaménko.

Interpretaci kladných i záporných čísel umožňuje tzv. doplňkový kód. Jeho princip je založen na redukci číselného obsahu datového formátu (byte, word . . .). Zatímco obsah přímého 8bitového kódu bez znaménka je 0 až 255, je doplňkový kód zobrazován v uspořádání podle obr. 59. Nejvyšší bit (b7) je vyhrazen pro znaménko, přičemž 0 = kladné, 1 = záporné číslo. Zbývajících sedm bitů umožňuje zapsat nejvyšší hodnotu kladného čísla 0111 1111 = 7H = 127 D. Kladné číslo lze převést na záporné dvěma způsoby, prostřednictvím prvního nebo druhého doplňku (complement).



Obr. 59. Struktura vyjádření binárního čísla v doplňkovém kódu

První doplněk binárního čísla se získá prostou inverzí, negací všech bitů binárního čísla.

Druhý doplněk se odvodí z prvního doplňku tak, že se k němu přičte 1 jako přenos do nejnižšího bitu.

Příklad:

číslo 6	0110
jeho 1. doplněk	1001
+	1
2. doplněk	1010

Odčítání se provádí přičtením druhého doplňku. Správnost lze ověřit na uvedeném příkladu

6	0110
+(—6)	1010
	1| 0000

Přenos ze znaménkového bitu se přitom zanedbává, výsledek je správný, roven nule.

Přehled o rozložení kladných a záporných čísel v 1. doplňku upřesňuje znázornění, využívající číselné osy (obr. 60). Zde má nula dvojí reprezentaci, kladnou (00 H) a zápornou (FF H). Proto je maximální číselný rozsah 1. doplňku —127 až + 127. Úpravou na druhý doplněk má nula jedinou hodnotu 00 H, číselný rozsah je roven —128 až +127.

Výsledkem operace odčítání může být i nula. To je další významný stav, který si označíme příznakem Z jako zero (nula).

Mechanismus technické realizace odčítání v doplňkovém kódu s klasickou binární sčítačkou je na obr. 61. První operand A je na sčítačku přiváděn v přímém tvaru, druhý operand B může být přiváděn buď v přímém nebo inverzním tvaru. Volbu umožňuje řídicí signál, ovládající funkci přístupového bloku buffer/invertorů. Tentýž signál zároveň ovládá logickou úroveň přenosového vstupu nejnižšího řádu sčítačky (c_0). Pokud je řízení nastaveno tak, že operand B prochází na vstup sčítačky v přímém tvaru a $c_0 = 0$, obvod aritmeticky sečte oba operandy, např.

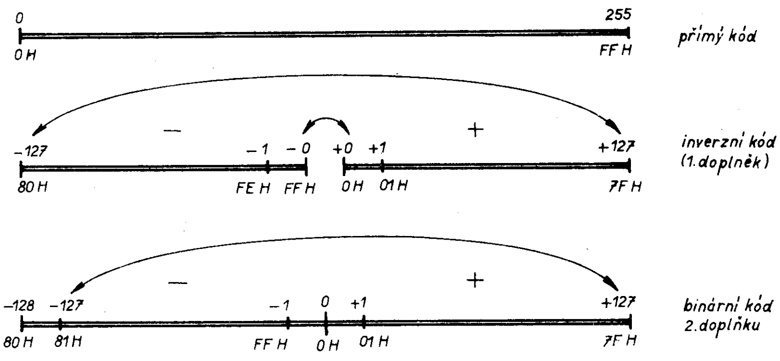
A	0011 0110	54
+B	0100 0101	+69
	0111 1011	123

Je-li naopak řízení nastaveno tak, aby byl operand B invertován, je vlastně na sčítačku přiváděn jeho 1. doplněk. Sčítačka tedy s menšitelem zachází jako s 1. doplňkem, ale současné nastavení přenosu $c_0 = 1$ inkrementuje výsledek a tím jej upravuje automaticky tak, jako by operace probíhala v 2. doplňku. Příklad

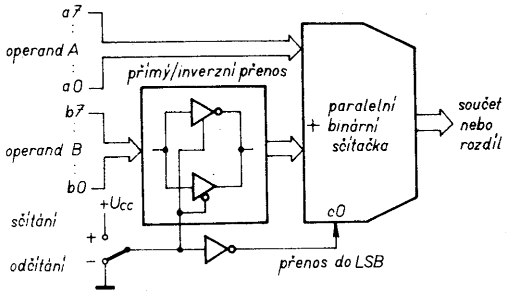
A	0011 0110	54
$\bar{B}$	1011 1010	+(—69)
	1111 0000	
+1	1	
Σ=	1111 0001	—15

Při slučování čísel v doplňkovém kódu je na rozdíl od využití CY v předchozí kapitole vyhodnocení přeplnění složitější. Lze vycházet ze skutečnosti, že přeplnění indikuje buď přenos z bitu nejvyššího významu (b6) do znaménkového bitu (b7), nebo ze znaménkového bitu (b7) do CY. Oba přenosy současně nastat nemohou.

Vidíme další potřebný indikátor, znaménko S (signum). Samozřejmě, že číselný rozsah výpočtu může být rozšířen prostřednictvím několikabytových operandů. Výpočty s vícenásobnou přesností jsou však již programovou záležitostí.



Obr. 60. Grafické porovnání 8bitového vyjádření a rozsahu binárního čísla v přímém, prvním a druhém doplňkovém kódu



Obr. 61. Princip využití binární sčítačky pro sčítání i odčítání v doplňkovém kódu

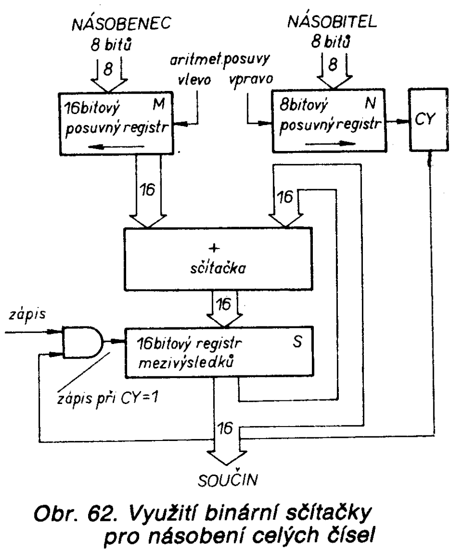
Násobení a dělení celých čísel

Běžné 8bitové mikroprocesory instrukcemi pro násobení a dělení vybaveny nejsou. Výjimku tvoří jednočipový mikropočítač řady 8051, kterého se snad již brzo dočkáme i u nás. V ostatních případech je nutno zmíněné operace zajišťovat programově. Možné principy mají opět analogii v dekadické soustavě. Nejjednodušší by bylo řešit násobení opakovaným sčítáním — např. součin 14×9 bychom získali devětkrát opakovaným přičtením čísla 14 do příslušného, předem vynulovaného registru např. akumulátoru. Základním nedostatkem je mimořádně dlouhá doba výpočtu.

Výhodnější je násobení, využívající aritmetických posuvů. Každý posuv operandu vlevo znamená jeho násobení dvěma. Příklad

```
  1011
 x 101
  1011   prvý zápis,  1011.1
 0000 ←  posuv vlevo, 1011.0,
                 tj. prázdný zápis
1011  ←  posuv vlevo, zápis 1011.1
110111   11.5=55
```

Na obr. 62 je příklad násobení čísel v přímém kódu, znovu založený na využití sčítačky. Předpokládejme, že oba 8bitové operandy jsou uloženy do posuvných registrů. Násobenec pro snazší pochopení do 16bitového, který bude posouván vlevo a násobitel do 8bitového. Ten bude naopak posouván vpravo, aby nejnižší bit vystupoval do přenosového indikátoru.



Obr. 62. Využití binární sčítačky pro násobení celých čísel

Násobením dvou 8bitových čísel můžeme získat 16bitový součin. Proto je užito 16bitového výsledkového registru, sloužícího zároveň jako registr dílčích součtů.

Funkce: Do nižšího bytu registru M zapíšeme násobenec, horní byte zůstává prázdný. Do registru N zapíšeme násobitel, výsledkový registr S vynulujeme. Z registru N posuvem vpravo vysuneme nejnižší bit (LSB). Je-li roven 1, zapíše se obsah registru M do registru S. Obsah M se pak posune vlevo a celý postup se stále opakuje, do registru S se přičítá nebo nepřičítá posouvaný násobenec tak dlouho, až je vyčerpán 8bitový formát násobitele, tj. počitadlo posuvů postupně dekrementovalo ze 7 na 0. Registr S, zachycující dílčí součty, v tom případě obsahuje výsledek.

Řešení na obr. 62 je pro snadné pochopení záměrně stylizováno nepříliš prakticky. Ve skutečnosti lze vystačit jak s 8bitovým registrem násobence, tak s 8bitovou sčítačkou. Pro násobitel i výsledek lze použít společný registrový pár, samozřejmě 16bitový.

Možný postup při dělení celých čísel je v podstatě opakem násobení. Dělenec lze obecně postupně zmenšovat opačnými posuvy a odčítáním dělitele tak, že po osmi posuvech je ve výsledkovém registru uložen celočíselný podíl, v jiném registru pak zbytek dělení. Zde již narážíme na problém přesnosti výpočtu.

Jiná číselná zobrazení

Jednou z možností, jak zvětšit rozlišovací schopnost číselné interpretace, je zavedení pevné „binární" čárky, obr. 63. Té se však využívá jen málo.

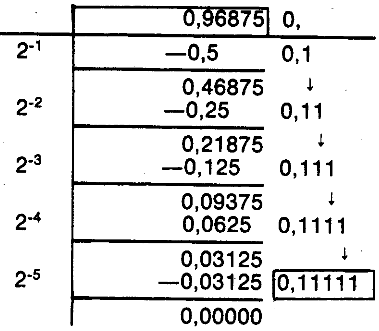
Současně zvětšit rozlišovací schopnost a zobrazovaný rozsah umožňuje číselná reprezentace s pohyblivou čárkou (floating point). Je známým případem vyjádření čísla v exponenciálním tvaru

$$y = a.b^x,$$

kde a = mantisa, b = základ, x = exponent.

K tomu je, podle požadovaného rozsahu, zapotřebí několik bytů. Příklad na obr. 64 užívá čtyřbytového vyjádření. Jeden z bytů slouží pro definici znaménka a celočíselného exponentu, zbývající tři pro zápis číselné mantisy. Základ b = 2. Mantisa je v daném případě normalizována tak, že se může pohybovat pouze v rozsahu $1/2 < a < 1$. Mechanismus převodu čísla do formátu floating point je jednoduchý. Pro určení mantisy se číslo dělí nejbližší větší mocninou dvou. Mantisa je určena podílem. Protože je vždy tvořena desetinným číslem, musí být nakonec převedena do tvaru čísla s pevnou binární čárkou.

Příklad: Mantisa čísla 15,5 se určí jako 15,5 : 16 = 0,96875 a platí
$15{,}5 = 0{,}96875 \cdot 2^4$.
Převod mantisy do binárního tvaru:



Tedy: $0{,}96875_{10} = 0{,}11111_2$.

Postup:

Je-li zbytek mantisy větší nebo roven menšiteli, zapisuje se do formátu pevné čárky 1, jinak 0. Iterace končí při nulovém výsledku, nebo vyčerpáním počtu míst mantisy. V tom případě je určení mantisy zatíženo chybou.

Plovoucí čárka je z praktického hlediska velmi výhodná. Její programové zajištění je ovšem značně náročné, z čehož vyplývá i základní nedostatek — dlouhá doba výpočtu. Ta se pochopitelně umocňuje při operacích s takto interpretovanými čísly. Je však třeba říci, že při většině technických aplikací se obvykle vystačí s matematickými operacemi s vícenásobnou přesností.

V mikropočítačích, které jsou určeny převážně pro matematické aplikace se zvětšenými nároky na rychlost, se užívá numerických koprocesorů. Ty spolupracují s hostitelským procesorem tak, že při provádění běžných instrukcí pouze sledují jeho činnost. Pomocí speciálních instrukcí v programu (např. ESC u 8086) poznává koprocesor (8087), že se dále bude jednat o aritmetické instrukce. Je-li instrukce taková, že bude vhodné využít možností koprocesoru, přechází pokračování programu na něj. Ukončení požadavku na spolupráci s koprocesorem se opět zapisuje do běžného programu speciální instrukcí. Numerické koprocesory, známé již dávno (např. APU 8231 pro CPU 8080), umožňují nejen rychlejší provádění aritmetických operací v různých datových formátech, ale i řady jiných funkcí (logaritmy, trigonometrické funkce . . .).

Úsporné vyjádření hodnoty binárního čísla

Zápis, čtení a vyhodnocení čísla ve formě jedniček a nul je zdlouhavé, nepohodlné a nepřehledné. Binární kódy a data, zadávaná mikropočítači nebo jím zprostředkovaná, musí být vyjadřovány jednodušší formou. Nejvhodnější pro tento účel je hexadecimální (šestnáctkový) kód. Jeho určení je jednoduché. Každý byte se rozdělí do dvou bitových čtveřic. Jednotlivým bitům každé čtveřice se přiřadí číselné váhy 8, 4, 2, 1. Výsledná číselná hodnota, vyjádřená bitovou čtveřicí, se proto může pohybovat pouze v rozsahu 0 až 15. Aby ji bylo možno úsporně vyjádřit jediným znakem, používají se pro rozsah 0 až 9 shodná čísla, pro rozsah 10 až 15 znaky A až F.

Příklad:

0000 0000 = 00 H
0111 1111 = 7F H
1000 0000 = 80 H

Obdobný je systém oktalového (osmičkového) kódu. Ten rozděluje každý byte na tři bitové skupiny, nejvyšší má pouze dva, zbývající vždy tři bity. Váhy jsou 4, 2, 1.

Příklad:

11 101 100 = 354 Q
01 011 100 = 134 Q.

Jistou předností oktalového kódu je to, že vystačí s pouze číselnými symboly 0 až 7. Pro interpretaci jednoho byte je však zapotřebí tří čísel.

Oba kódy jsou uspořádány v tabulce na obr. 65.

Dekad. číslo	Hexa	Octal	BCD-packed	Gray
0	0	0	0000 0000	0000
1	1	1	0000 0001	0001
2	2	2	0000 0010	0011
3	3	3	0000 0011	0010
4	4	4	0000 0100	0110
5	5	5	0000 0101	0111
6	6	6	0000 0110	0101
7	7	7	0000 0111	0100
8	8	10	0000 1000	1100
9	9	11	0000 1001	1101
10	A	12	0001 0000	1111
11	B	13	0001 0001	1110
12	C	14	0001 0010	1010
13	D	15	0001 0011	1011
14	E	16	0001 0100	1001
15	F	17	0001 0101	1000

Obr. 65. Srovnávací tabulka mnemonických kódů hexa-oktal, BCD-Gray

Číselné kódy

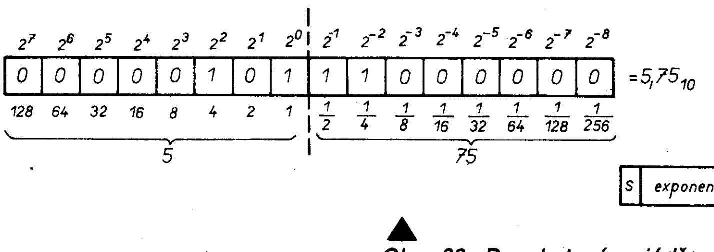
Kromě binárního existuje celá řada dalších číselných kódů. Mnohé z nich přímo nesouvisí s vlastní činností mikropočítače. Mají však v určitých oblastech použití některé výhodné vlastnosti.

Kód BCD

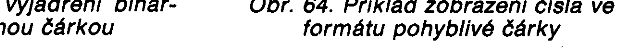
Kódu BCD přísluší mezi ostatními prioritní postavení. Umožňuje binárně kódovaný dekadický zápis a tím usnadňuje převody mezi „člověčí" dekadickou a počítačovou binární soustavou. Lze vidět formální podobnost s kódem hexa — každá číselná pozice je vyjádřena čtveřicí bitů. Jejich váhový součet se pohybuje v rozsahu 0 až 9, viz obr. 65.

V jednom bytu tedy mohou být vyjádřena dvě dekadická čísla. Pak mluvíme o zhuštěném (packed) kódu BCD s maximální zobrazitelnou hodnotu 99_{10}. Někdy se užívá i nezhuštěného kódu BCD, kdy obsahem celého bytu je jediné číslo BCD.

Mikroprocesory ovšem přímo v kódu BCD pracovat nedokáží. Veškeré operace, tedy i operace s čísly BCD

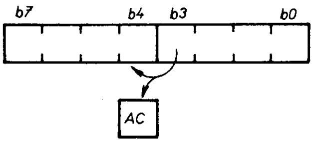


Obr. 63. Dvoubytové vyjádření binárního čísla s pevnou čárkou



Obr. 64. Příklad zobrazení čísla ve formátu pohyblivé čárky

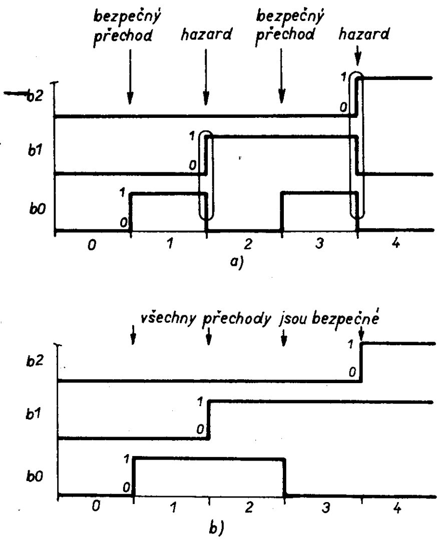
probíhají v binárním kódu. Procesorové jednotky jsou proto vybaveny prostředky, umožňujícími potřebné korekce výsledků — příklad využití instrukce DAA je uveden dále. Zde si pouze v této souvislosti všimneme dalšího důležitého příznaku AC — auxiliary carry (pomocného přenosu). Označuje a indikuje se jím přenos mezi bitovými čtveřicemi (čísly BCD) v rámci jednoho bytu, obr. 66.



Obr. 66. Přenos z nižší (b3) do vyšší (b4) tetrády bytu indikuje příznak AC

Grayův kód

I když se dalšími kódy zabývat nemůžeme, učiňme výjimku alespoň u Grayova kódu, užívaného často zvláště u rotačních nebo délkových snímačů polohy. Pro tento účel lze těžko použít běžný binární kód, protože u něj může při průchodech mezi sousedními číselnými kódy, kdy se najednou mění úrovně několika bitů, docházet vlivem mechanického kmitání a nestabilit snímače k chybnému vyhodnocení (obr. 67a). Grayův kód je uspořádán tak, aby se mezi sousedními číselnými stavy mohla měnit pouze úroveň jediného bitu (obr. 67b). Pak je z tohoto hlediska kód bezpečný.



Obr. 67. Porovnání binárního a Grayova kódu; a) binární kód je zdrojem hazardních stavů, mění-li se úroveň většího počtu než jednoho bitu, b) Grayův kód je charakteristický bezpečným přechodem do sousední číselné hodnoty

K dalším známým 4bitovým kódům patří zejména Aikenův kód 2421, kódy BCD, +3, Gray + 3 nebo kód 8-4-2-1.

Pro konverzi desítkových čísel se někdy s výhodou užívá i 5bitových kódů. Z nich jsou nejznámější Johnsonův kód a kód 2 z 5.

Kód ASCII

Při komunikaci mikropočítače s vnějším prostředím (periferiemi) zpravidla nelze používat přímo binární kód. Uvažme např. pouze periferie typu klávesnice, displej, tiskárna... Jsou znakově orientovány. Číslice není číselná hodnota, je to znak, který této hodnotě nějakým způsobem odpovídá. Stejný význam má v dané situaci i písmeno nebo funkční operátor. Mikropočítač však musí zajišťovat i jiné funkce, než je příjem nebo vyslání nějakého znaku. Musí ovládat činnost a snímat funkční stavy všech periferií, které jsou vůči němu ve vztahu podřízených, vlastně pasívních členů.

Aby se určitým způsobem standardizovaly funkce zařízení různých výrobců, využívá se pro tyto komunikace národních nebo firemních modifikací kódu ASCII — American Standard Code for Information Interchange. U nás platný kód ISO 7 je na obr. 68. Obsah kódu můžeme rozdělit do dvou skupin. První tvoří služební a řídicí znaky, vyhrazené pro řízení komunikací s různými typy

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	¤	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	FM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	∧	n	—
F	SI	US	/	?	O	_	o	DEL

příklad použití:
A = 41 H
0 = 30 H

Obr. 68. Tabulka znakových a řídicích kódů ISO 7 ≐ ASCII

periferií. Druhá skupina je tvořena grafickými znaky, tj. písmeny, číslicemi, symboly a funkčními operátory. Číslovky i písmena jsou uspořádány tak, že tvoří vždy souvislou množinu, posloupnost kódů. Číslovkám odpovídají pro 0 až 9 kódy 30 až 39 H, písmenům A až Z kódy 41 až 5A H a malým písmenům a až z kódy 61 až 7A H. Toho se užívá při vzájemných konverzích bin ↔ ASCII zavedením příslušných posuvů, např. 1 B = 1 H + 30 H = 1 ASCII.

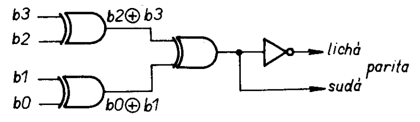
Jak vidíme z obr. 68, vlastní kód ASCII je pouze 7bitový, má rozsah 0 H až 7F H. Osmý bit proto může být využit pro paritní zabezpečení, nebo pro specifické rozšíření kódu, například národní abecedou apod.

Kódové zabezpečení

Zvláště při již naznačených přenosech nebo ukládání datových bloků mohou uplatněním různých vlivů (rušení, nespolehlivost záznamového média..) vznikat chyby. Proto se v praxi užívá různých kontrolních opatření, zabezpečovacích a samoopravných kódů. To je opět záležitost, které se můžeme dotknout pouze v náznaku.

Kontrola paritou

Je to nejjednodušší a proto i nejpoužívanější, ale současně i nejméně dokonalý způsob kontroly. Princip parity (rovnosti, správnosti) je založen na tom, že se užitečný datový formát doplňuje dalším, nadbytečným (redundantním) bitem. Tento paritní bit pak již tvoří součást kódu, i když sám hodnotovou informaci neobsahuje. Logická hodnota paritního bitu je v podstatě volitelná. Může být nastavena tak, aby výsledná parita byla buď lichá (celý formát včetně P obsahuje lichý počet logických jedniček), nebo sudá. To se zajišťuje v některých případech programovým nastavením. Úroveň paritního bitu však vyhodnocuje poměrně jednoduchý kombinační obvod. Příklad řešení s hradly EXOR pro 4bitový užitečný formát je na obr. 69.



Obr. 69. Generátor paritního bitu s hradly EXOR

Příznak parity P je dalším důležitým indikátorem, užívaným v mikropočítačové technice.

Testem správnosti nebo chyby paritního bitu lze s poměrně velkou pravděpodobností usuzovat na to, zda při zpracování nebo přenosu vznikla chyba. Metoda je ovšem založena na předpokladu, že se chyba vyskytne pouze v jediném bytu. To však nikdy není jisté. Obsah přenášeného formátu může být zkomolen i když počet jedniček zůstane stejný. To pak paritní kontrola nerozezná. Situaci nejlépe postihuje jednoduchý příklad kontroly 2bitového kódu, který může mít čtyři užitečné kombinace:

Data	Lichá parita
0 0	1
0 1	0
1 0	0
1 1	1

Již na tomto jednoduchém příkladu vidíme, že rozšíření 2bitového datového kódu na 3bitový redundantní kód pro zcela bezpečnou kontrolu nestačí. Změní-li se současně oba bity původních dat, parita tuto chybu nerozezná.

Kontrolní součet

Ke zvětšení bezpečnosti se při přenosech datových bloků často využívá programového zabezpečení tzv. kontrolním součtem (Check Sum). Jednotlivé byty celého bloku se sčítají tak, že se uvažuje pouze nižší část výsledku, přenos mimo rámec bytu se zanedbává. Realizace je tedy poměrně jednoduchá a časově nepříliš náročná. Výsledkem, kontrolním součtem, se příslušný datový blok doplní. Při kontrole se stejným způsobem opět vypočte nový kontrolní součet a porovnává s původním. Při jejich shodě se přenos, zabezpečovaný navíc paritou, považuje za úspěšný. Parita v tomto případě kontroluje řádky, kontrolní součet sloupce datového bloku.

Hammingův kód

Vychází ze stejného principu jako paritní kontrola. Bezpečnost kódu je touto cestou možno zvětšit pouze rozšířením počtu redundantních bitů v samotném kódu. Hammingův kód a jeho modifikace využívají takové struktury a rozložení paritních bitů, které umožňují chyby nejen bezpečně zjistit, ale i lokalizovat.

Kódy jsou vytvářeny tak, že každý z redundantních bitů kontroluje paritu určité váhové kombinace. Kombinovaným vyhodnocením jednotlivých parit pak lze získat informaci o tom, zda a ve kterém bitu nebo bitech (včetně redundantních) se chyba vyskytla. Pokud se vyskytla pouze jediná chyba, dává identifikace chybného bitu možnost chybu automaticky odstranit. Výskyt dvojnásobné chyby lze pouze indikovat. Obvody, které umožňují chybu opravit, jsou již ovšem velmi složité. Proto se prakticky využívají pouze jako součást speciálních obvodů, např. řadičů diskových pamětí.

Kódy CRC

Zvláštní a účinnou detekci vzniklých chyb, včetně chybových shluků, umožňuje technika cyklických redundantních kontrol, CRC. Protože je založena na hardwarovém principu, umožňuje současně rychlé přenosy. Proto se užívá především v řadičích diskových pamětí, ale i např. v telekomunikacích.

Princip lze velmi zhruba přirovnat ke kontrolnímu součtu. V tomto případě se však nesaldují jednotlivé byty přenášeného bloku, ale naopak různé bity, vytvářené hardwarovým obvodem. Ty mají k obsahu přenášeného bloku nepřímý, matematicky vyjádřený vztah. Představme si, že hodnotu přenášeného bloku vyjádříme jako určitý, předem neznámý mnohočlen P. Budeme-li jej naopak dělit druhým, tentokrát přesně známým a definovaným mnohočlenem Q, vznikne určitý, pro nás dále nezajímavý podíl Y a zbytek Z. Každý datový blok v takovém případě bude popsán jemu přesně odpovídajícím zbytkem Z. Tento zbytek představuje redundantní kód CRC a je příznakem obsahu příslušného datového bloku.

Odpovídající obvodová struktura, která tvorbu bloku CRC může zabezpečit, je blokově znázorněna na obr. 70. Realizuje funkci dělicího mnohočlenu Q. Tento polynom má nejčastěji normalizovaný formát CCIT = $1+x^5+x^{12}+x^{16}$, užívají se však i jiné. Zapojení tuto funkci realizuje speciálním posuvným registrem, jehož dílčí přímé a zpětné vazby zajišťují součtové obvody typu EXOR. Jestliže se na vstup registru přivede užitečný datový signál v sériovém tvaru a odstartuje se zápis uvolněním přístupového hradla a hodinového signálu, mění se stav registru podle obsahu vstupního signálu, který je současně zapisován do paměťového média. Bude-li současně s ukončením přenosu zablokován přístup k registru, zůstává v něm zbytek dělení, rovný CRC. Tím se pak doplní datový blok na disku tak, že se obsah registru vysune přes příslušný výstup. Při kontrole správnosti zapsaných dat, již opatřených příslušným kódem CRC, přes stejný obvod, získává řadič možnost detekovat výskyt chyb porovnáním obsahu původního a nově generovaného kódu CRC. Vyskytne-li se chyba, řídicí software uskutečňuje další pokusy. Teprve pokud se po určitém počtu pokusů nesetká s úspěchem, hlásí tuto chybu operátorovi.

II. Typická struktura a činnost univerzálního mikroprocesoru

Dost dlouho jsme váhali, jaký konkrétní typ mikroprocesoru zvolit pro úvod do principů činnosti a obvodového řešení. Za hlavní cíl jsme si vytkli co možno nejsrozumitelnější popis, aby bylo možno snadno pochopit základy struktury univerzální CPU. Pod tímto zorným úhlem jsme jako nejvhodnější nakonec zvolili klasickou jednotku CPU 8080A. To, co je při její praktické aplikaci největší slabinou, tj. potřeba podpůrných obvodů, se naopak při rozboru funkce ukazuje být předností, celková struktura mikroprocesoru je otevřenější a umožňuje snadněji pochopit jednotlivé souvislosti.

CPU 8080A

CPU 8080A je 8bitová dynamická programovatelná jednotka, pracující v binárním kódu, se základní dobou taktu 0,5 µs, která dobře vyhovuje přístupovým dobám běžných pamětí.

Aby každý mikroprocesor, který je sám o sobě pouhou součástkou, mohl vůbec pracovat, musí být vždy doplněn do sestavy mikropočítače, tedy alespoň obvody operační paměti (ROM, RAM) a vstupů/výstupů.

CPU 8080A není úplný mikroprocesor. Pro tuto funkci musí být doplněn obvody generátoru hodinového signálu 8224 a externím systémovým řadičem 8228, popř. 8238 pro systémy s rozsáhlejší sběrnicí.

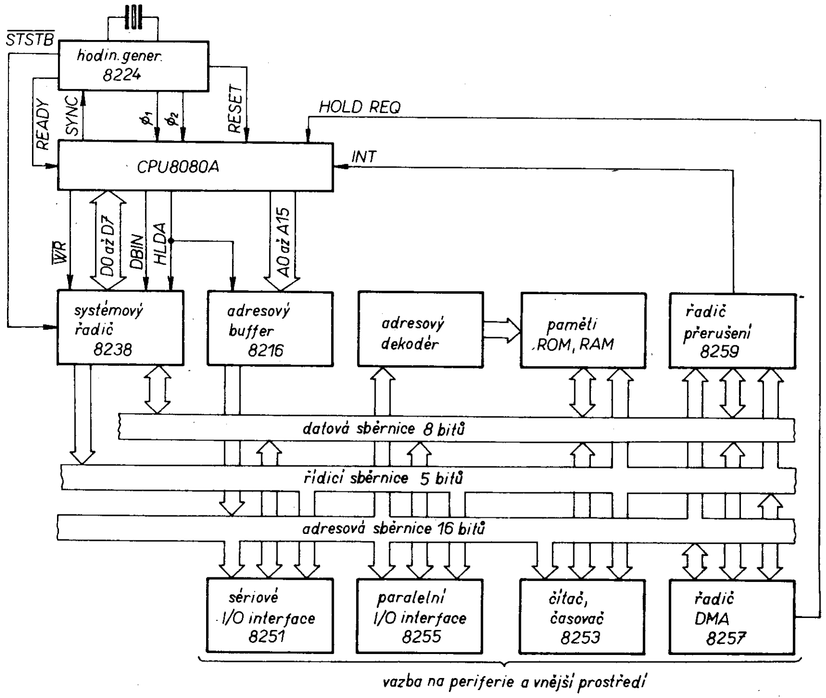
Mikropočítač s CPU 8080A je obecně znázorněn na obr. 71. Jeho efektivní výstavbu umožňují speciální doplňkové obvody, odpovídající řadě MCS 80. CPU 8080A je mikroprogramovaná jednotka. Znamená to, jak jsme již dříve uvedli, že provedení instrukce a tím i její výkonnost nejsou omezeny dobou trvání taktu hodinového synchronizačního signálu. Naopak, k provedení každé instrukce je zapotřebí vykonat řadu mikrooperací. Doba trvání instrukce (instrukční cyklus) je proto proměnná, závisí na typu instrukce, někdy i na vnějších podmínkách.

Obecný přehled o průběhu instrukčního cyklu je na obr. 72. Instrukční cyklus se skládá ze strojových cyklů M1 až M5. Jejich počet, ale i typy pochopitelně opět v první řadě závisí na typu instrukce.

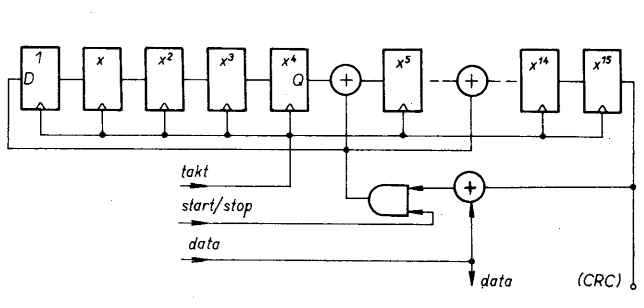
Každý ze strojových cyklů lze opět rozložit na doby T_1 až T_5, jejichž trvání je pevně určeno taktem hodinového signálu, počet dob pak opět závisí na typu instrukce a případné vnější podmínce (T_W-wait).

Konečně každá doba se skládá ze dvou fází, první z nich je detekční (ϕ_1), druhá výkonná (ϕ_2).

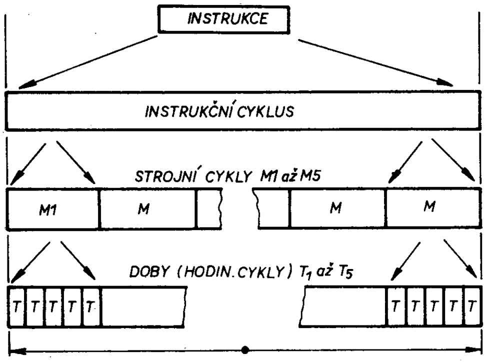
Skutečný průběh realizace každé instrukce je určen především jejím operačním kódem. Instrukce CPU 8080A mají podle typu a způsobu adresování různé formáty, jsou jedno, dvou a tříbytové, příklady viz obr. 73. Operační kód instrukce je obsažen vždy v prvním bytu. Jak patrno z obrázku, v tomto bytu mohou být



Obr. 71. Přehledové blokové schéma mikropočítače s CPU 8080A, podpůrnými a doplňkovými obvody řady MCS80



Obr. 70. Posuvný registr tvorby kódu CRC



Obr. 72. Grafické znázornění skladby instrukčního cyklu

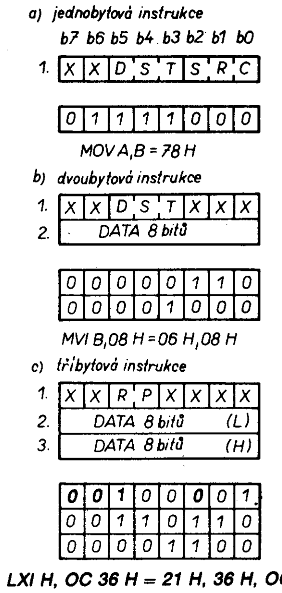
uvedeny i odkazy na případné operandy (SRC — zdrojový registr, DST — cílový registr, RP — registrový pár). U tříbytových instrukcí obsahuje druhý byte vždy nižší, třetí byte vyšší část adresy nebo operandu.

V závislosti na obsahu operačního kódu je řízena celá posloupnost ve všech strojních cyklech, které jsou podle operačního kódu vybírány a řazeny.

Dále si popíšeme strukturu CPU podle hrubého blokového schématu na obr. 74. Vazba CPU na vlastní doplňkové obvody je zajišťována paralelními systémovými sběrnicemi, 8bitovou obousměrnou datovou sběrnicí D0 až D7 a adresovou 16bitovou sběrnicí A0 až A15. Řídicí sběrnice, jak vidíme, chybí. Zajišťuje ji systémový externí řadič, ke kterému se dostaneme později.

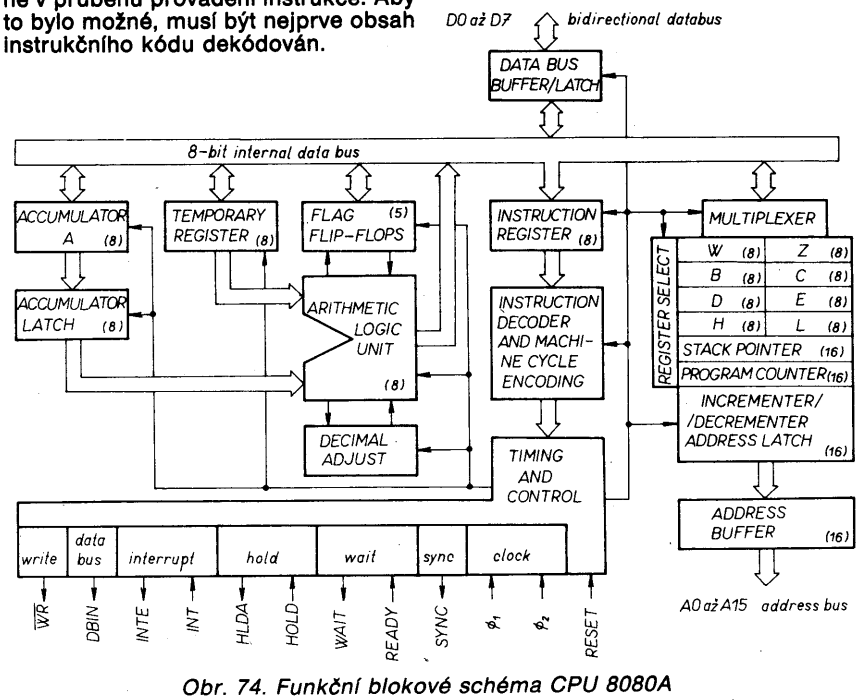
Vnitřní struktura CPU

Abychom zdůraznili jeho význam, začneme netypicky registrem instrukcí, IR. Je to paralelní 8bitový registr, do kterého se vždy na počátku instrukčního cyklu zapisuje operační kód aktuální instrukce, právě čtené z operační paměti mikropočítače. Je důležité si uvědomit, že operační kód zůstává v IR po celou dobu trvání instrukčního cyklu. Obsahem operačního kódu instrukce, zapsané v IR, jsou řízeny veškeré operace a komunikace potřebné v průběhu provádění instrukce. Aby to bylo možné, musí být nejprve obsah instrukčního kódu dekódován.

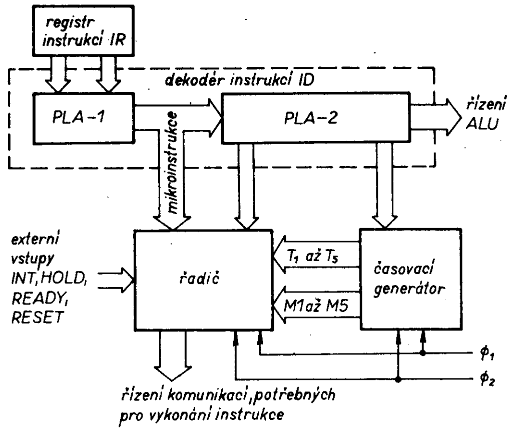


Obr. 73. Příklady různých formátů instrukcí; a) i v jednoduchém formátu může instrukční pole obsahovat odkazy na případné operandy, viz obecný i konkrétní příklad, instrukce typu MOV R, R s registrovým adresováním, b) 2bytová instrukce s odkazem na první operand (DST) v instrukčním poli, druhý operand je specifikován ve druhém bytu instrukce typu MVI R, DATA 8 s registrovým adresováním, c) 3bytová instrukce s implicitním adresováním cílového registru v instrukčním poli a přímým operandem ve zbývajících bytech instrukce

Dekodér instrukcí, ID, je u 8080A tvořen dvojúrovňovým logickým polem, jehož funkce je závislá na obsahu IR, obr. 75. Výstupy pole 1. úrovně se označují jako mikroinstrukce. Jsou to aktivované kódy, ovládající jednak činnost interního řadiče, jednak pole 2. úrovně. To pak řídí průběh instrukčního cyklu (řazení strojových cyklů M1 až M5 a jejich dob T_1 až T_5) a funkce aritmeticko/logické jednotky.



Obr. 74. Funkční blokové schéma CPU 8080A



Obr. 75. Dekodér instrukcí v systému řadiče

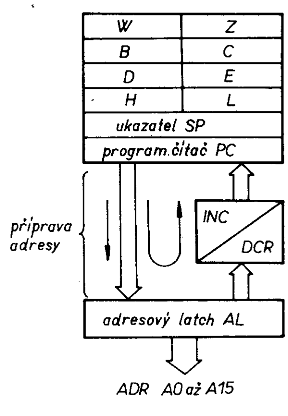
Bloku interního řadiče pak již nezbývá nic jiného, než chovat se pro každou instrukci jako specializovaný automat, ovládající veškeré činnosti, potřebné k vykonání požadované operace. Navíc musí reagovat na některé speciální externí signály.

Abychom celý tento blok mohli považovat za systémový řadič (budeme-li externí obvod 8228 považovat za jeho součást), musíme do něj zahrnout i obvody, zajišťující adresování a výběr jednotlivých bytů aktuálních instrukcí a dat z operační paměti.

Adresa platného paměťového místa je na systémovou sběrnici vysílána prostřednictvím 16bitového adresového bufferu, schopného tedy přímo adresovat rozsah 0 až 65 535 bytů. Buffer je buzen adresovým latchem, AL, obr. 76. Způsoby, jakými se v AL vytváří platná adresa, jsou však velmi rozmanité, závislé na typu instrukce.

Při běžném, řetězeném adresování podle krokové konvence dodává platnou adresu do AL 16bitový programový čítač PC. Jeho stav upravuje blok INC/DCR, řízený řadičem. V daném případě v průběhu každého bytu instrukce inkrementuje obsah PC a tak mu připravuje novou adresu.

Blok INC/DCR však nemusí spolupracovat pouze s čítačem PC. V zásadě je vázán pouze na adresový latch AL. Mimo PC má přístup také k ostatním registrům, především k dočasnému registrovému páru WZ, páru HL a ukazateli zásobníku SP. Jejich stav může inkrementovat, dekrementovat,



Obr. 76. Detail k popisu adresování různých typů instrukcí

nebo do nich přepisovat původní obsah adresového latche AL. Tyto funkce jsou nezbytné vzhledem k užívaným adresovacím metodám (viz např. registrové adresování), funkci a mechanismu zásobníkové paměti, instrukcím skoků, volání a návratů z podprogramů nebo přerušení.

Pro určitou orientaci alespoň několik příkladů. Instrukce přímého skoku vyžaduje změnit adresu příští instrukce na základě hodnoty, obsažené v 2bytovém operandu. Tato adresa se nejprve zapíše do dočasného páru WZ a teprve odtud se přepíše přímo do adresového latche AL. Pak již, inkrementovaným přepisem obsahu AL do programového čítače, může adresování pokračovat normálně prostřednictvím PC. U instrukcí podmíněných skoků je adresovací mechanismus obdobný s tím rozdílem, že není-li podmínka skoku splněna, obsah páru WZ se nepoužije a čítač PC, v průběhu čtení instrukce podmíněného skoku běžně inkrementovaný, normálně adresuje následující instrukci.

U instrukce volání podprogramu (CALL) je navíc nutno odložit návratovou adresu, tj. obsah již inkrementovaného čítače PC, do zásobníku. Adresování u této instrukce probíhá tak, že se nejprve dočasně „uklidí" cílová adresa skoku do podprogramu, opět do registrového páru WZ. Následuje přesun návratové adresy z PC do zásobníku. Za pomoci bloku INC/DCR se nejprve dekrementuje ukazatel zásobníku SP, aby ukazoval na následující zapisovanou položku, aktuální vrchol zásobníku. Ukazatel se pak přepíše do AL a tak, přes sběrnici A0 až A15, fyzicky adresuje místo zásobníkové paměti, kde bude uložen vyšší byte PC (zásobník je vratný), tedy návratové adresy. Opakovaným adresovacím postupem, tedy další dekrementací SP a přesunem do AL se uloží do zásobníku i nižší byte čítače PC, popř. návratové adresy. Teprve nyní může být počáteční adresa podprogramu přepsána z WZ do adresového latche AL a adresovací mechanismus může přejít do normálního režimu ve volaném podprogramu.

Návrat z podprogramu (RET) se uskutečňuje znovu s pomocí páru WZ. Nejprve se z běžného vrcholu zásobníku čte nižší, po inkrementaci SP vyšší byte návratové adresy. Opakovanou inkrementací se aktualizuje vrchol zásobníku a může být přepsána návratová adresa z WZ do AL.

Ponecháme zatím řadič stranou a přejdeme k operační jednotce. Jejím jádrem je ALU, aritmeticko/logická jednotka, jejímž ústředním obvodem je modifikovaná struktura binární sčítačky. Základní funkce akumulátoru (ACC i ACT) i registru TMP jsme si již vysvětlili. Prakticky zbývá pouze doplnit, že oba dočasné registry, tj. ACT a TMP jsou programově nepřístupné. Používá je, podobně jako registry W a Z, samotný systém CPU k přechodnému ukládání dat. Oba operandy mohou být prostřednictvím registrů ACC a TMP přiváděny k ALU přes datovou interní sběrnici jak z paměti, tak z libovolného zápisníkového registru nebo vstupu.

Modifikovaná funkce sčítačky, řízená dekodérem instrukcí, dovoluje ALU provádět základní aritmetické operace v binární soustavě (součet, rozdíl, komparace), logické operace a rotace. Pro korekci výsledků binárních operací s čísly v kódu BCD je ALU vybavena obvody dekadické korekce, DAA.

Nedílnou a mimořádně důležitou součástí ALU je registr příznakových indikátorů F, který se skládá ze samostatných klopných obvodů, registrujících některé mezní výsledky a příznaky matematických a logických operací, které jsme si již definovali. Formát příznakového registru je na obr. 77. Těchto příznaků využívá jako podmínek ke svému provedení celá řada podmíněných instrukcí, které jsou základem systému větvení programu.

Významnou součástí operační jednotky je i blok zápisníkových registrů. CPU 8080A je registrově orientovaná jednotka. Disponuje skupinou šesti nezávislých 8bitových registrů B, C, D, E, H, L, které mohou být využívány pomocí příslušných instrukcí i jako trojice 16bitových registrových párů BC, DE, HL. To omezuje potřebu odkládat operandy a dílčí výsledky do operační paměti, což by při adresových

b7	*b6*	*b5*	*b4*	*b3*	*b2*	*b1*	*b0*
S	Z	0	AC	0	P	1	CY
SIGNUM	ZERO		AUXILIARY CARRY		PARITY		CARRY

Obr. 77. Formát příznakového registru; nastavení jednotlivých příznakových indikátorů definuje:
SIGNUM, znaménko
záporný výsledek operace
ZERO, nula nulový výsledek operace
AUX. CARRY, pomocný přenos
přenos mezi nižší a vyšší čtveřicí bitů stradače
PARITY, parita
příznak sudé parity
CARRY, přenos
příznak přenosu z nejvyššího bitu

schopnostech 8080A vyžadovalo značné časové nároky.

V souvislosti s dočasným registrem TMP je dobré vědět, že se do něj, současně se zápisem do instrukčního registru, zapisuje i operační kód instrukce. Toho se využívá při zpracování žádosti o přerušení. S úlohou programově nepřístupných registrů W, jsme se již seznámili. Komunikaci 16bitových registrovaných párů prostřednictvím 8bitové interní datové sběrnice umožňuje multiplexer MUX. Tím je 16bitové číslo rozdělováno na dvě poloviny, přičemž registrům B/C, D/E, H/L vždy přísluší vyšší/nižší byte.

Registrový pár HL má kromě své univerzální funkce také speciální určení. Může sloužit jako ukazatel fiktivního registru, paměťového místa v operační paměti. V souvislosti s aritmetickou instrukcí DAD pro sčítání s dvojnásobnou přesností se pár HL používá jako 16bitový střadač.

16bitový ukazatel zásobníku SP jsme již uvedli. Přesto bude vhodné si jeho smyslu a funkce všimnout ještě jednou. Jeho hlavní úlohou je umožnit vytvoření zásobníkové paměti s takovým přístupem, jaký je typický pro paměť LIFO (obr. 30c), přímo v normální operační paměti. Taková paměť se vždy musí chovat tak, jako by měla jedinou přístupovou cestu. Musí umožňovat zapsat celou řadu dat, přitom však může být čtena vždy pouze posledně zapsaná položka — a na tu právě musí ukazovat SP. Poslední zapsané položce říkáme vrchol zásobníku. Ukazatel SP se nastavuje pouze jednou, když se instrukcí SPHL nastaví na požadované adrese dno zásobníku. Pak se již příslušnými instrukcemi přístupu k zásobníku (typicky PUSH/POP, CALL/RET) ukazatel SP nastavuje automaticky tak, aby stále ukazoval na vrchol zásobníku.

Z předchozího vyplývá, že zapíšeme-li do zásobníku libovolnou posloupnost bytů, můžeme je vybírat pouze v opačném pořadí, než v jakém byly vloženy. To naprosto přesně vyhovuje potřebnému mechanismu ukládání a vybírání návratových adres podprogramů a to i jak jednoduchých, tak hnízděných — tak se označuje případ, kdy dochází k řetězenému volání jednoho podprogramu druhým, až nakonec, jsou-li všechny provedeny, následuje návrat do hlavního, běžného programu (princip je na obr. 78).

Nakonec zdůrazněme důsledek toho, že zásobník je vratný — při zápisu se vrchol zásobníku pohybuje od jeho dna směrem k nižším adresám (ukazatel se tedy dekrementuje), při čtení, tedy vybírání ze zásobníku, adresa roste, protože ukazatel je inkrementován.

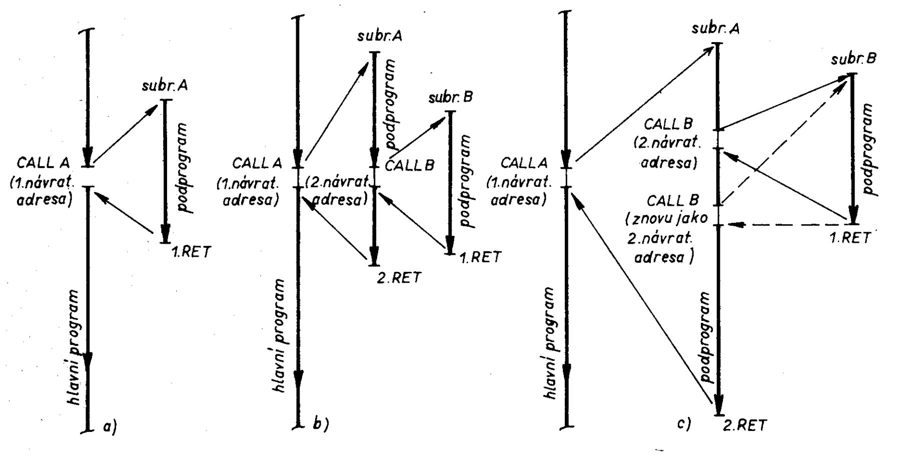
Vraťme se opět k řadiči, jehož struktura není úplná. Chybí mu obvody generátoru hodinového signálu a řízení korespondence s vnějšími obvody (mimo CPU), operační pamětí a obvody I/O. Funkci příslušných podpůrných obvodů si hned stručně projdeme. Předtím si však ještě všimněme skupiny speciálních signálů, které tvoří vstupy/výstupy bloku časování CPU.

Kromě vstupů hodinového signálu fází ϕ_1, ϕ_2 zde nacházíme především signál RESET. Jeho funkce je jasná, zajišťuje definovaný start mikropočítače tak, že impulsem, generovaným buď ručně nebo automaticky při přivedení napájecího napětí, se nuluje programový čítač PC. To má mimořádně významný důsledek. Provádění programu začíná na jeho nulté adrese. Přesně tam musí být uložena první instrukce programu. Jedině tak lze zajistit, že řadič pozná, zda právě čte operační kód instrukce nebo nějaké binární číslo. Jakmile je však zajištěno, že první byte, který po startu zapsal instrukční registr, je skutečně operační kód instrukce, je už další činnost bezpečná. V opačném případě by se program zhroutil. RESET nenastavuje registry CPU, což je výhodné zvláště při opakovaných startech. Při prvním zapnutí jsou stavy registrů nedefinované.

Signál SYNC je časovacím blokem generován na počátku každého strojního cyklu, označuje dobu T_1. Jeho význam dále uvidíme. Zbývají čtyři signálové dvojice.

Dvojice READY/WAIT umožňuje zavést synchronizaci mezi činností CPU a pomalou pamětí nebo periferií. Signálem READY = L může být CPU žádána o čekání — pak vkládá do strojního cyklu čekací doby T_W a v činnosti nepokračuje, což hlásí výstupním signálem WAIT. Celého mechanismu se také využívá při oživování systému, protože umožňuje procházet programem po jednotlivých strojních cyklech.

Vstupním signálem HOLD = H může být CPU uvedena do pasívního režimu. Její datová a adresová sběrnice přechází do 3. stavu. Tak se řeší režim přímého vstupu do paměti DMA.



Obr. 78. Mechanismus zásobníkové paměti umožňuje jednoduché, hnízděné a rekurzívní volání podprogramů; a) jednoduché volání, b) volání a návraty vhnízděných (nesting) podprogramů, c) složitější systém hnízdění podprogramů (s rekurzí)

Příjem požadavku HOLD je potvrzován signálem HLDA.

Mimořádný význam má signál INT, odpovídající vstupu žádosti o přerušení. Přijetí žádosti je možno buď programově povolit (instrukcí EI), nebo zakázat (DI). Tím se nastavuje nebo nuluje vnitřní klopný obvod INTE. Za předpokladu, že je žádost o přerušení povolena (INTE = H), je po aktivaci vstupu INT přerušen běžící program způsobem, který připomíná volání podprogramu.

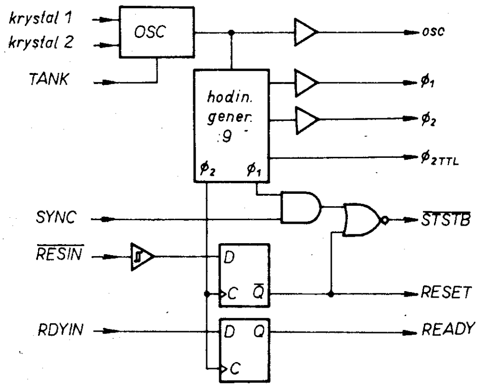
Signál DBIN indikuje, že probíhá čtení do CPU. Signál $\overline{WR}$ naopak řídí zápis směrem z CPU. Není rozlišeno, zda se jedná o komunikaci paměťovou nebo I/O.

Podpůrné obvody

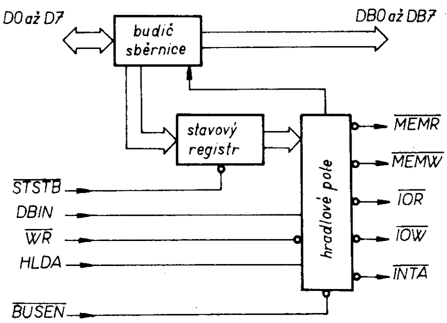
Obvod 8224, viz obr. 79, obsahuje obvody oscilátoru, řízeného krystalem. Základní kmitočet f_{osc} je dělen v poměru 1:9 na dvojici nepřekrývajících se hodinových signálů „fází" ϕ_1, ϕ_2.

Od nich jsou odvozeny doby a fáze činnosti CPU. V závislosti na vstupních asynchronních signálech $\overline{RESIN}$, RDYIN a signálu SYNC, který je výstupem CPU, identifikujícím začátek každého strojového cyklu, generuje 8224 již zmíněné synchronizované signály pro nastavení počátečního stavu RESET, pro případné vložení čekací doby T_W, READY, a vzorkování stavového slova, $\overline{STSTB}$ — Status Strobe. Poslední signál si dobře pamatujte.

Obvod 8228 (popř. 8238 pro systémy s rozsáhlejšími sběrnicemi — odlišné časování řídicích výstupů) především



Obr. 79. Blokové schéma generátoru hodinového signálu a synchronizovaného budiče 8224



Obr. 80. Blokové schéma systémového řadiče a budiče datové sběrnice, 8228/8238

doplňuje strukturu řadiče o blok tvorby výkonových signálů řídicí a datové sběrnice (obr. 80). Obvod současně umožňuje, spolu s dalšími externími obvody, zavést poměrně univerzálně zpracování žádosti o přerušení programu (interrupt).

Pro orientaci v činnosti celé sestavy základní procesorové jednotky, tvořené CPU 8080A, generátorem 8224 a řadičem 8228 (obr. 81) je nutné pamatovat si stále dvě základní fakta:

1. Operační kód, určující zpracování každé instrukce, je po celou dobu trvání jejího instrukčního cyklu uložen v instrukčním registru IR, tedy v CPU.
2. Aktuální stav při zpracování této instrukce popisuje, rozloženě pro každý její strojový cyklus, tzv. stavové slovo — Status Word, opět po celou dobu trvání strojového cyklu zapsané ve stavovém registru řadiče 8228.

Struktura řadiče obsahuje kromě obousměrného budiče datové sběrnice především stavový registr a dekodér řídicí sběrnice. Vazbu mezi CPU a 8228 zajišťuje obousměrná sběrnice D0 až D7, řídicí signály DBIN, $\overline{WR}$, HLDA a strobovací signál $\overline{STSTB}$. Asynchronní signál BUSEN umožňuje uvést výstupy řadiče 8228 do 3. stavu.

Stavové slovo

Stavové slovo vysílá CPU po datové sběrnici do řadiče 8228 v době T_1 až T_2 na začátku každého strojového cyklu. V tomto časovém intervalu není systémová datová sběrnice DB používána pro žádnou komunikaci (obr. 82). Stavovým slovem je identifikován typ právě začínajícího strojového cyklu. Byla-li např. právě ukončena instrukce, nebo stimulován start systému inicializací vstupu $\overline{RESET}$, CPU stavovým slovem oznamuje, že v začínajícím cyklu bude číst operační kód instrukce (1, 2 nebo 3bytové). Je-li však již instrukce zpracovávána, status vždy identifikuje právě aktuální typ cyklu. Podle toho pak dekodér řadiče 8228 generuje takové signály řídicí systémové sběrnice, které zajistí požadované komunikace mezi CPU, operační pamětí a obvody I/O, nebo ošetření speciálních stavů.

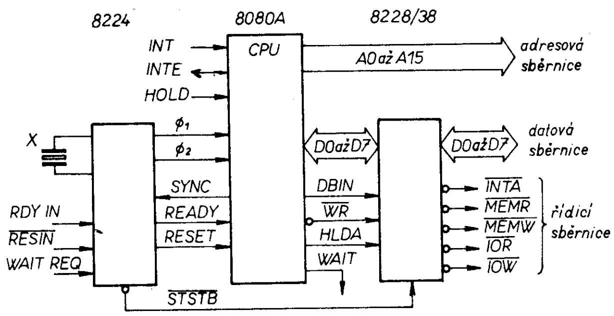
Stavové slovo se skládá z osmi bitů:

stavový bit		význam
INTA	D0	CPU potvrzuje přijetí požadavku na interrupt
$\overline{WO}$	D1	CPU bude zapisovat na datovou sběrnici
STACK	D2	příznak práce se zásobníkem
HLTA	D3	CPU se nachází ve stavu HALT
OUT	D4	zápis na výstupní port
M1	D5	příznak čtení 1. bytu instrukce
INP	D6	čtení ze vstupního portu
MEMR	D7	příznak čtení z paměťové oblasti

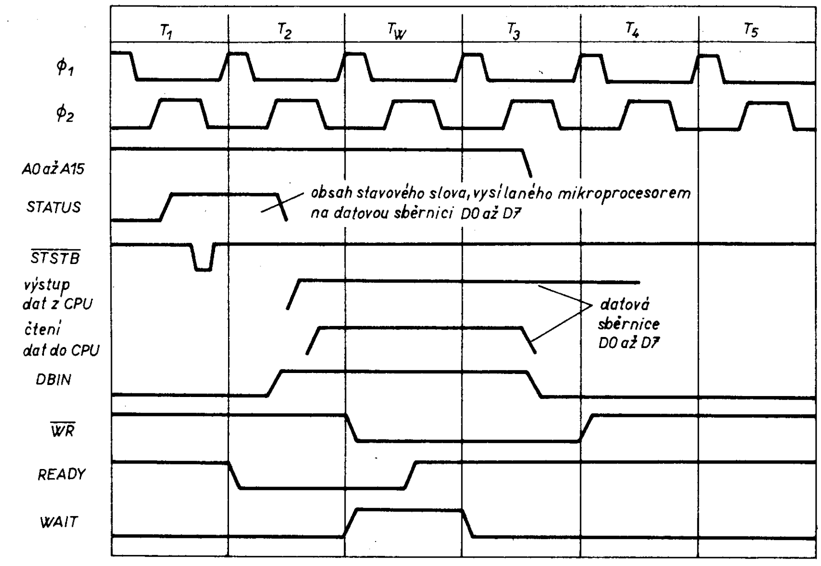
Významy jednotlivých bitů jsou vesměs snadno pochopitelné, bližší pozornost budeme později věnovat především signálům INTA a HLTA.

CPU poskytuje celkem deset možných bitových kombinací, tím i deset možných typů stavového slova a tedy i strojového cyklu (obr. 83).

Jestliže přemýšlíme, proč je to vše zařízeno právě uvedeným způsobem, je třeba uvážit dobu vzniku CPU 8080. Pak hned vidíme celou řadu důvodů — generátor 8224 i řadič 8228 musely být vzhledem k tehdejším technologickým možnostem řešeny jako bipolární, tedy externí obvody. Tím se zvětšoval počet potřebných vývodů pouzdra CPU. Využití společné „interní" datové sběrnice D0 až D7 pro vazbu CPU na řadič i systémovou sběrnici řešilo mnohé problémy. Navíc bylo možno (analýzou stavového slova dostupného na



Obr. 81. Mikroprocesorová sestava CPU8080, generátor 8224, řadič sběrnice 8228



Obr. 82. Časový diagram průběhu zobecněného strojního cyklu; T_1 — aktivace adresové sběrnice A0 až A15 pro komunikaci s pamětí nebo I/O, vyslání stavového slova na D0 až D7, T_2 — test signálů READY, HOLD a HLTA, inkrementace programového čítače PC, T_W — možné vložení čekacích dob WAIT jako důsledek předchozích testů, T_3 — ve strojním cyklu M1 čtení operačního kódu instrukce, jinak čtení dalších slabik instrukce nebo čtení/zápis dat, T_4, T_5 — tyto doby se používají pouze u instrukcí a cyklů, které je potřebují ke svému dokončení

CPU prostřednictvím signálů DBIN (čtení do CPU — $\overline{I/OR}$, $\overline{MEMR}$, $\overline{INTA}$) a $\overline{WR}$ (výstup z CPU — $\overline{I/OW}$, $\overline{MEMW}$).

V době T_2, která vždy nepodmíněně následuje po T_1, se testují signály READY, HOLD, HLTA. Potom se již ve fázi ϕ_2 této doby inkrementuje programový čítač PC a tím s předstihem připravuje následující běžná adresa. Ze stavu T_2 se na základě předchozího textu modifikuje průběh strojového cyklu, popř. se větví stavový diagram, obr. 84. Nepřehlédněme dále, že zatímco READY a HOLD jsou odezvou na vnější signály, HLTA je potvrzením instrukce HLT.

Pokud je READY=H, HLTA=L, pak se při přechodu z T_2 může uplatnit pouze externí požadavek HOLD = H na uvedení CPU do pasívního stavu při současném odpojení sběrnic. Toho se využívá tehdy, žádají-li spolupracující zařízení, řadič nebo jiná jednotka multiprocesorového systému o přístup ke společné operační paměti bez účasti CPU (prostřednictvím obvodu přímého přístupu k paměti DMA).

Externí žádost HOLD je obecně asynchronní. V CPU je proto nejprve synchronizována nastavením interního klopného obvodu HOLD F/F. Je-li žádost zachycena s dostatečným časovým přestihem, přechází CPU do 3. stavu v době T_3/ϕ_1 libovolného cyklu a

Obr. 83. Tabulka stavových slov jednotlivých strojních cyklů

		1	2	3	4	5	6	7	8	9	10
		FETCH	MEMORY		STACK		I/O		INTERRUPT	HALT	
BIT	STATUS	čtení instr.	čtení	zápis	čtení	zápis	čtení	zápis	akcept.	akcept.	akcept. INT
D0	INTA	L	L	L	L	L	L	L	H	L	H
D1	$\overline{WO}$	H	H	L	H	L	H	L	H	H	H
D2	STACK	L	L	L	H	H	L	L	L	L	L
D3	HLTA	L	L	L	L	L	L	L	L	H	H
D4	OUT	L	L	L	L	L	L	H	L	L	L
D5	M1	H	L	L	L	L	L	L	H	L	H
D6	INP	L	L	L	L	L	H	L	L	L	L
D7	MEMR	H	H	L	H	L	L	L	L	L	L

vývodech) získat řadu cenných informací o chování systému při ladění nebo opravách. To byly a stále jsou cenné atributy. I když dnes jsou již zkušenosti i měřicí technika na jiné úrovni, stále se využívá nejrůznějších cest ke zmenšení počtu vývodů a zajištění efektivní diagnostiky.

Časování a stavy CPU

Jednotlivé instrukce, popř. instrukční cykly se skládají z různého počtu strojových cyklů. Ty zase obsahují různé počty dob. Jaké typy strojových cyklů (M1 až M10), v jaké posloupnosti (M1 až M5) a kolik jich je v instrukčním cyklu uplatněno, závisí na typu instrukce. V podstatě platí, že cyklus M1 může obsahovat různý počet dob (T_1 až T_3, T_4, T_5), cykly M2 až M5 pak zpravidla pouze doby T_1 až T_3.

Cyklus M1 (Fetch) se vždy užívá ke čtení operačního kódu instrukce. Někdy, u instrukcí, které nevyžadují přenos po systémové datové sběrnici, stačí cyklus M1 pro vykonání celé instrukce, např. přesunové MOV r,r. U několikabytových instrukcí se užívají cykly M2 a M3 ke čtení dalších dat, viz např. zápis přímého 8bitového MVI r, data nebo 16bitového LXI RP, data operandu. Cykly M4 a M5 pak mohou být využity pro libovolný směr přenosu po datové sběrnici.

Při startu systému přes vstup RESET začíná doba T_1 prvního strojového cyklu adresováním první instrukce, uložené na nulté adrese v operační paměti počítače. Prvním cyklem je M1 (Fetch), čtení operačního kódu instrukce.

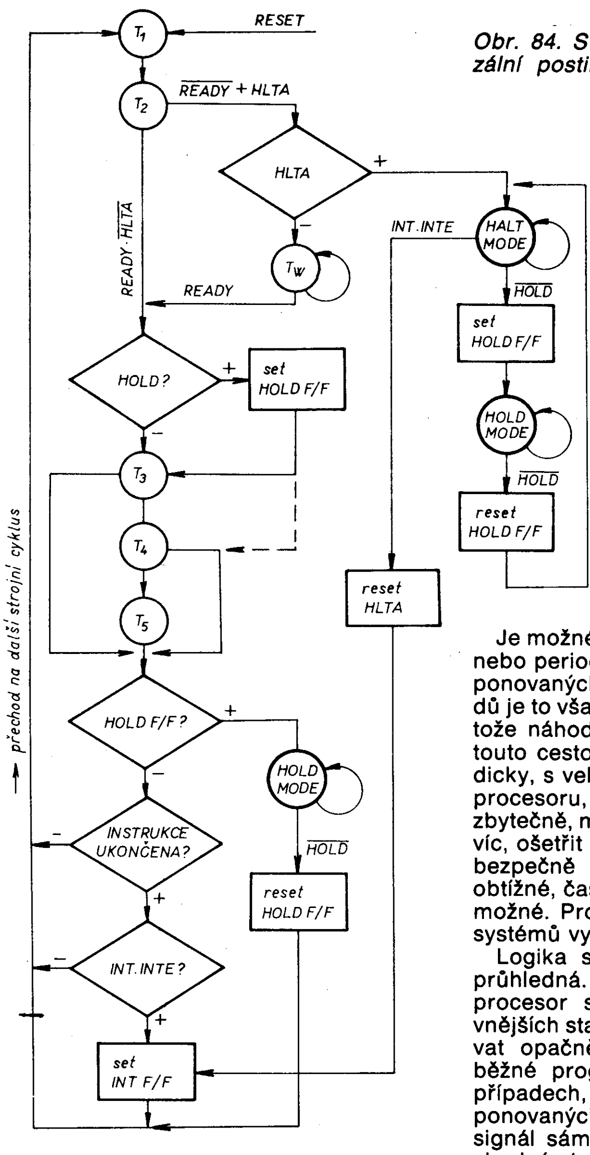
V obecném případě však obdobně můžeme uvažovat průběh jakéhokoli strojového cyklu, každý bude začínat dobou T_1. Obecný průběh takového cyklu budeme sledovat jednak na časovém diagramu (obr. 82), jednak současně na stavovém diagramu (obr. 84). Začínáme v době T_1, kdy CPU vysílá na adresovou sběrnici A0 až A15 adresu paměťové nebo I/O lokace, podle typu cyklu. Jeho identifikaci vysílá CPU ve formě stavového slova po datové sběrnici D0 až D7 do řadiče 8228 jen s nepatrným zpožděním za adresou, s fází ϕ_2 doby T_1. V tomto intervalu je systémová datová sběrnice odpojena, nemůže nastat žádná kolize. Stavové slovo se přepisuje přes datový budič do stavového registru 8228 strobovacím impulsem $\overline{STSTB}$, vhodně odvozeným od impulsu SYNC v obvodu generátoru 8224. Ve stavovém registru je slovo uloženo po celou dobu trvání strojového cyklu. Proto během jeho trvání může nastat jen jeden přenos po systémové datové sběrnici. Vlastní časování signálů řídicí sběrnice ovládá

nastaví potvrzující výstupní bitový signál HLDA = H. Jinak se může stát, že do stavu HOLD přejde CPU až po době T_3, viz vývojový diagram. V dobách T_4, T_5 může CPU dokončit probíhající strojový cyklus.

Ukončení stavu HOLD je v tomto případě zcela logické, nastane ihned, jakmile je zrušen externí požadavek HOLD, kterému opět odpovídá i synchronizované nulování interního návěští HOLD F/F. V době T_1 následujícího strojového cyklu přechází CPU do aktivního režimu, tj. pokračuje v původním programu, což indikuje i nulování výstupu HLDA. Je třeba si pamatovat, že se požadavkem HOLD sice „přeruší" prováděný program, ale až do opětovného spuštění CPU žádnou jinou činnost nevykonává. Není to tedy skutečné přerušení, ale pouze zastavení činnosti, během něhož je CPU dokonale pasívní. Proto mj. v tomto stavu také není schopna reagovat na případnou žádost o skutečné přerušení programu.

Neuplatní-li se při přechodu z doby T_2 READY, HLTA ani HOLD, přechází CPU přímo do doby T_3. Je-li naopak aktivní pouze žádost READY = L, pak interní řadič CPU vkládá místo doby T_3 pasívní dobu T_W tak dlouho, dokud nebude READY = H. Touto cestou se dosahuje synchronizace CPU s pomalými pamětmi. Pak se automaticky pokračuje v průběhu strojového cyklu dobou T_3.

Doba T_3 je vlastně tou „hlavní" do-



Obr. 84. Stavový diagram pro univerzální postih průběhu strojního cyklu a módů CPU

bou každého cyklu. V prvním cyklu M1 každé instrukce se teprve nyní čte její operační kód z paměťového místa, adresovaného již v době T_1 stejného cyklu. Tento z dynamických důvodů (přístupové doby pamětí) nutný časový předstih znamená, že se v době T_3 začne skutečně realizovat nová instrukce.

V dalších strojových cyklech se již doby T_3 využívá jinak, k již zmíněnému čtení dalších bytů instrukce nebo čtení/zápisu dat.

Pokračujme v průběhu strojového cyklu tam, kde jsme přestali, tj. po průchodu dobami T_1 až T_5. Pokud se v průběhu cyklu neuplatnil požadavek HOLD a nejedná se o poslední cyklus celé instrukce, přejde mikroprocesor do doby T_1 dalšího strojového cyklu. Jedná-li se však současně o ukončení celé instrukce, testuje se v této konečné fázi, byl-li v průběhu realizace instrukčního cyklu uplatněn požadavek přerušení probíhajícího programu.

Přerušení probíhajícího programu (interrupt)

Problematika a význam možnosti přerušit běžící program zasluhuje podrobnější vysvětlení. Interrupt v podstatě řeší jednu ze systémových slabin programovatelného automatu, který by jinak, na rozdíl od svého hardwarového protějšku, mohl na vnější události reagovat pouze sledováním stavu vybraných proměnných, sledovaných prostřednictvím vstupních (input) obvodů. Tato metoda také většině praktických požadavků vyhovuje. S výjimkou těch, jejichž ošetření je naléhavé a nesnese odkladu.

Je možné a často se užívá cyklického nebo periodického sledování stavu exponovaných periferií. Ve většině případů je to však velmi neekonomické, protože náhodná žádost o ošetření bude touto cestou identifikována jen sporadicky, s velmi malou výtěžností aktivity procesoru, který naopak vždy, většinou zbytečně, musí odložit jiné činnosti. Navíc, ošetřit tímto způsobem efektivně a bezpečně několik periferií je značně obtížné, často z časových důvodů i nemožné. Proto se u mikropočítačových systémů využívá metody přerušení.

Logika systému přerušení je zcela průhledná. Proč neustále zatěžovat procesor sledováním řady kritických vnějších stavů, když je možno postupovat opačně. Nechat systém, ať řeší běžné programové úlohy a pouze v případech, kdy některá z vybraných exponovaných periferií nebo speciální signál sám požádá o ošetření, zajistit vhodnými, zpravidla technickými i programovými prostředky jejich obsluhu. Potom se, v nejjednodušším případě, může procesor vrátit k původnímu programu až do doby, kdy bude uplatněna další taková žádost. Celý mechanismus spočívá tedy v tom, že se v důsledku uplatnění vhodné žádosti přeruší běžící program, žádosti se programově ošetří a pak se pokračuje v přerušeném programu. Vidíme, že touto cestou se z činnosti ztrácí doba čekání na vnější událost, i když se ovšem při vzájemných přechodech mezi běžícím programem a ošetřením přerušení vždy nějaká časová ztráta vyskytne. Ztráty vyplývají především z potřeby ukládat a vyzvedávat potřebná data a parametry. To je ovšem vzhledem k předchozímu příkladu v běžných situacích zanedbatelné.

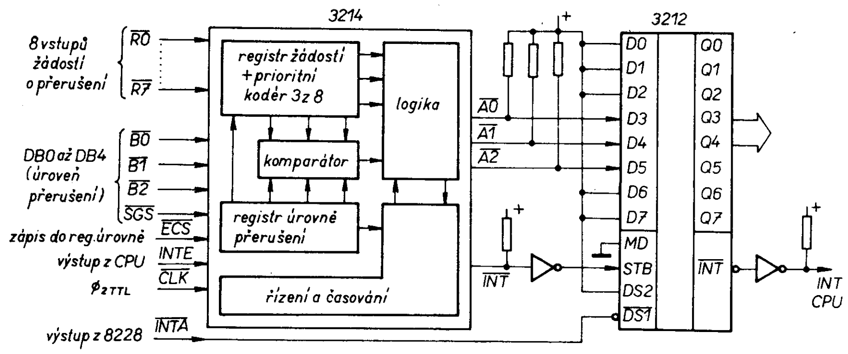
Výhody metody přerušení programu vyniknou ještě zřetelněji, uvážíme-li, že periferií a signálů, vyžadujících okamžitého ošetření, může být a bývá několik. Navíc je často mezi nimi třeba uplatnit různé, mnohdy proměnné vzájemné priority nebo naopak blokovat některé žádosti. Pokud vám celý princip přerušení (interrupt) poněkud připomíná techniku volání a návratů z podprogramů, jsme doma. Hlavní a zásadní rozdíl je v tom, že přerušení není voláno programově, ale technickými prostředky. Proto ani počáteční adresa obsluhy přerušení (obdoba cílové adresy skoku do podprogramu, ekvivalent CALL apod.) nemůže být čtena z operační paměti, ale je generována technickými prostředky tak, jako by ji dodávala sama periferie, žádající o přerušení.

Princip generování počáteční adresy přerušení je jednoduchý. Je založen na tzv. vektorovém adresování, kterého jsme si již všimli v souvislosti s logickým procesorem MC14500, u něhož ovšem nebylo dotaženo do důsledku. Dále uvažovaný systém umožňuje výběr z několika pevných adres na počátku operační paměti, určených zkrácenými 8bitovými instrukcemi typu RST *n*.

Zopakujme, že požadavek na přerušení se vlastní jednotce CPU předává nastavením vstupu INT jejího interního řadiče na úroveň H. CPU pak musí být předána technická adresa počátku programově zajištěné obsluhy přerušení. Analogie s voláním podprogramu napovídá, že před přechodem k obsluze musí být uklizena návratová adresa běžícího programu. Pro její určení musí být nejprve dokončen celý instrukční cyklus, v jehož průběhu byl uplatněn požadavek. To v diagramu na obr. 84 postihuje test dokončení instrukce, po němž teprve následuje test, zda byl požadavek skutečně uplatněn. CPU 8080A disponuje možností přerušení programově povolit (instrukce EI) nebo zakázat (instrukce DI), což je základem výstavby složitějšího, tzv. prioritního systému přerušení.

Obecně asynchronní požadavek na přerušení, INT, je synchronizován interním klopným obvodem INT F/F ve struktuře CPU. Je-li povoleno přerušení, indikuje to stav bitového výstupu CPU INTE = H. Pak při platné žádosti o přerušení je logický součin INT . INTE = H a INT F/F = H. Z diagramu vidíme, že se přechází do nového, tentokrát speciálního strojového cyklu M1-I (Interrupt), který je určen právě nastavením příznaku přerušení INT F/F. Jeho průběh je obdobou cyklu M1-Fetch s tím rozdílem, že ve stavovém slově je místo signálu MEMR (D7 — čtení z paměti) generován speciální signál INTA (D0). V řadiči CPU se navíc neinkrementuje PC. Nuluje se signál INTE a tím je zakázáno jakékoli další přerušení. Stavové slovo je dekódováno řadičem 8228 a projeví se tak, že v době T_3 cyklu M1-I není na datové sběrnici CPU aktivován obsah žádného paměťového místa. Tím je externím obvodům přerušení umožněno, aby (v nejjednodušší verzi) v tomto časovém intervalu dodaly na datovou sběrnici kód s formátem RST *n*, tj. 11XX X111, kde XXX představuje vektor adresy přerušení. Tak je možno adresovat jednu z možných restartových adres RST 0 až 7, fyzikálně 0, 8, 16 až 56 D, popř. 0, 8, 10 až 38 H. Současně se ukládá návratová adresa do zásobníku (stack).

Zajímavý, prakticky využitelný případ představuje instrukce RST 7 = 1111 1111. V případě potřeby pouze jediné úrovně přerušení stačí výstup $\overline{INTA}$ systémového řadiče 8228 spojit přes rezistor s napětím +12 V. Sama vnitřní struktura řadiče pak při potvrzení přerušení impulsem INTA vysílá na datovou sběrnici CPU kód vektoru RST 7 (nejsou třeba žádné další externí obvody). Této metody se někdy užívá v rozšířené verzi s programovou identifikací zdroje přerušení.



Obr. 85. Princip prioritního vektorového systému přerušení s řadičem 3214

Běžné praktické aplikace ovšem vyžadují doplnit mikropočítač externími obvody, technickými prostředky systému přerušení. Stylizované řešení s dnes již jen výjimečně užívaným řadičem 8úrovňového prioritního přerušení na obr. 85 volíme opět pouze z důvodu snadného pochopení podstaty.

Řadič 3214 se skládá zhruba z pěti podstatných funkčních bloků. 8bitový registr žádostí o přerušení (úrovňových) ovládá prioritní kodér, vyhodnocující vždy v 3bitovém binárním kódu aktuální žádost nejvyšší úrovně. Ta je porovnávána obvodem komparátoru priority úrovní s hodnotou, programově zapsanou do registru běžící úrovně přerušení. Předpokládejme, že odpovídá úrovni právě obsluhované žádosti. Do registru úrovně se zapisuje signálem $\overline{ECS}$. Signál INTE, obvykle odvozovaný od stejnojmenného výstupu CPU, umožňuje žádost o přerušení jakékoli úrovně blokovat. Hodinový signál $\overline{CLK}$, využívající fáze ϕ_{2TTL} z obvodu 8224, synchronizuje výstup $\overline{INT}$, po úpravě ovládající přerušovací vstup CPU. 3stavové výstupy A0 až A2 bloku logiky 3214 představují kód vektoru přerušení. Ostatní řídicí signály řadiče jsme zanedbali.

Praktické využití řadiče je podmíněno jeho doplněním obvodem 3212, zapojeným jako strobovaný výstupní latch s potvrzením uskutečněného zápisu dat jeho interním klopným obvodem SR. Jakmile řadič 3214 přijme žádost o přerušení, generuje na svém synchronizačním výstupu impuls H→L→H, užitý po inverzi jako STB pro zápis do interního latche 3212. Tím je do latche přepsán nejen vektor přerušení z řadiče, ale celá, externím zapojením vstupů 3212 doplněná adresa RST *n*. Současně je s týlovou hranou STB nastaven výstup $\overline{INT}$ 3212 na úroveň L, která po inverzi představuje aktivaci přerušení vlastního CPU. Tím začíná již známý proces, nastaví se interní obvod INT F/F a je vyvolán cyklus M1-I. Obsah stavového slova vyvolá vznik impulsu $\overline{INTA}$ = L na řídicí sběrnici a proto se v době T_3 přepíše obsah latche 3212 na datovou sběrnici CPU. Začíná vlastní ošetření přerušení vyvoláním příslušné subrutiny. Další požadavek může být uplatněn až po opětovném povolení přerušení instrukcí EI a mimo to také po opětovném zápisu do registru úrovně 3214.

Dále se již touto problematikou, která však zasluhuje skutečně podrobného studia, zabývat nemůžeme. Dnes téměř výlučně užívaný řadič 8259A je již mnohem rafinovanější programovatelný obvod, umožňující dynamické programování módů přerušení, maskování, volbu mezi přerušovacím a dotazovacím (polled) režimem a navíc, ve spolupráci s řadičem 8228, i aktivaci podprogramu z libovolné paměťové oblasti.

Zastavení programu instrukcí HLT

Vraťme se k diagramu na obr. 84. Při jeho větvení ve stavu T_2 jsme dosud neuvažovali instrukci HLT. Je to jednobytová, dvoucyklová instrukce, ovlivňující bit HLTA stavového slova. Okamžitě po jejím zpracování přechází CPU do stavu HALT, označovaného jako zastavení programu. Je to vlastně obdoba stavu WAIT, ovšem s odlišným průběhem ukončení.

Diagram ukazuje, že ze stavu HALT vedou dvě cesty. Logicky vzato existuje ještě jedna, vlastně havarijní — v každém případě lze využít vstupu RESET k opětovnému startu systému. V případě, že před instrukcí HLT nebylo povoleno přerušení, to bývá jediná, poslední možnost.

Také první z naznačených cest, kterou představuje externí vstup HOLD, není praktická. Jak vidíme z diagramu, odpovídající nastavení interního klopného obvodu HOLD F/F má sice za následek přechod do módu HOLD, po zrušení externího požadavku a tím nulování příznaku HOLD F/F však procesor opět přechází do módu HALT.

Jediné praktické využití módu HALT představuje opět přerušení. Kvůli němu je také instrukce HLT zařazena do instrukčního souboru. Jak vidíme z diagramu, při výskytu žádosti o přerušení na vstupu CPU je stav HALT okamžitě zrušen, nuluje se bit HLTA stavového slova a CPU přechází na obsluhu přerušení.

Prošli jsme poměrně podrobně vnitřní strukturu i funkční principy jednoduchého univerzálního mikroprocesoru, jehož je sestava CPU 8080 a odpovídajících podpůrných obvodů typickým představitelem. Všechny základní principy, kterými jsme se zabývali, mají všeobecnou platnost. Lze jich využít, ať se již budeme v budoucnu zabývat jakýmikoli typy mikroprocesorů nebo jednočipových mikropočítačů. Mikroprocesor ovšem není jedinou součástí mikropočítače. Užívá se celé řady dalších doplňkových obvodů, dnes již většinou programovatelných. Jejich složitost si se samotným mikroprocesorem mnohdy v ničem nezadá. Vždy je však mezi nimi jeden podstatný rozdíl: doplňkové obvody jsou vzhledem k CPU svým způsobem pasívními, podřízenými prvky mikropočítačové sestavy. Protože jsou obvykle „specializovány" (vykonávají určitou, omezenou skupinu funkcí) jsou vždy „přehlednější" a jejich činnost lze snáze zvládnout, než je tomu v případě CPU. Konstruktér i programátor ovšem musí možnosti a specifika těchto obvodů znát, respektovat je i dokázat jich využít.

Sestava mikropočítače

Možnost využití programovatelných obvodů ze stavebnicové řady MCS-80 je zřejmá z blokového schématu na obr. 71. Vlastním jádrem mikropočítače je blok mikroprocesoru (CPU 8080, generátor 8224 a řadič sběrnice 8228) spolu s obvody operační paměti (EPROM, RAM). Doplňkové obvody zajišťují především komunikaci s vnějším prostředím (paralelní I/O obvod 8255, obvod USART pro synchronní/asynchronní sériový přenos 8251), ale i další důležité funkce (obvod čítačů/časovačů 8253, řadič přerušení 8259, nebo přímého přístupu k paměti DMA 8257).

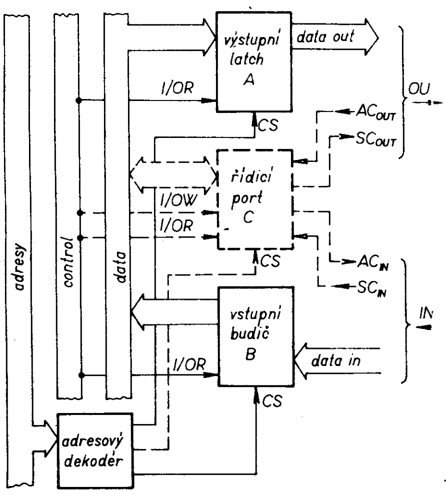
Všechny tyto obvody komunikují s CPU přes systémovou sběrnici (8bitový data bus, 16bitový address bus a 5bitový control bus). Všechny obvody včetně CPU jsou na obousměrnou datovou sběrnici vázány ve smyslu zdroj—cíl datového přenosu přes interní 3stavové buffery. Jednotlivé obvody jsou v průběhu sběrnicového cyklu aktivovány řadičem podle běžící instrukce. Jejich výběr zajišťuje systémový adresový dekodér. Ten definuje především adresový prostor operační paměti, ke kterému zajišťují směr přístupu signály $\overline{MEMR}$, $\overline{MEMW}$ a prostor vstupů/výstupů, řízený obdobně signály řídicí sběrnice $\overline{I/OR}$, $\overline{I/OW}$.

Paralelní synchronní vstup/výstup dat

Pro vazbu mikropočítače na vnější prostředí mají rozhodující význam interfaceové obvody I/O (Input/Output — vstup/výstup). Nejčastěji se užívá přenosu dat v paralelním stavu. Pro řízení vstupu a výstupu dat disponuje instrukční soubor CPU 8080 pouze dvěma, střadačově orientovanými instrukcemi IN a OUT. Instrukcí IN je do střadače CPU přenášen obsah systémové datové sběrnice, instrukcí OUT je naopak obsah ACC přenášen na datovou sběrnici. IN a OUT jsou vždy specifikovány přímou adresou portu I/O, určenou druhým bytem instrukce. Tato adresa se vysílá z CPU po nižším bytu adresové sběrnice. Jejím dekódováním lze rozlišit 2^8=256 vstupních a výstupních periférií.

Nejjednodušší technické řešení přenosu dat mezi ACC a portem I/O je na obr. 86. Neuvažujeme zatím obvody, zakreslené čárkovaně. Data, vysílaná z CPU instrukcí OUT, jsou zapisována do příslušného výstupního latche. Jeho výběr zajišťuje dekodér platné adresy, zápis do latche je strobován signálem I/OW. Data, která naopak mají být čtena instrukcí IN, mohou být na datovou sběrnici přiváděna přes jednoduchý 3stavový oddělovací budič za předpokladu, že po dobu přenosu budou stabilní. Aktivaci vstupního budiče zajišťuje opět adresový dekodér a řídicí signál I/OR.

Použitelnost naznačené metody je ovšem velmi omezená. Data, vysílaná instrukcí OUT, musí být kvůli zpracování držena na výstupu latche A tak



Obr. 86. K rozboru vazby CPU na paralelní porty I/O

dlouho, dokud nejsou přepsána další instrukcí OUT. Takové použití je možné při jednoduché aktivaci různých indikátorů nebo ovládacích členů (LED, číslicovka, relé, motor...) Vstupní data jsou omezena stejným způsobem, navíc musí být zajištěna jejich stabilita při čtení. Při přenosech datového bloku oběma směry však neexistuje žádná přímá možnost identifikace, o jaká data se v přenášené posloupnosti vlastně jedná. Obě komunikace probíhají i bez jakéhokoli potvrzení nebo kontroly přenosu.

Literatura

[1] *Jiřina, M.; Kottek, E.: Krátký, V.:* Navrhování číslicových obvodů. SNTL: Praha 1988.
[2] *Bernard, J. M.; Hugon, J.; Corvec, R.:* Od logických obvodů k mikroprocesorům. SNTL: Praha 1982.
[3] *Kokeš, J.:* Konečný automat. AR 10/88.
[4] *Kollert, E.:* Výpočetní technika. SNTL: Praha 1987.
[5] *Starý, J.:* Mikropočítač a jeho programování. SNTL: Praha 1988.
[6] *Kroha, P.; Slavík, P.:* Basic pro začátečníky. SNTL: Praha 1988.
[7] *Bayer, J.:* Mikroprocesory — architektura a pomocné obvody. DT ČSVTS, Ústí n.L. 1981.
[8] *Dědina, B.; Valášek, P.:* Mikroprocesory a mikropočítače. SNTL: Praha 1983.
[9] *Černoch, M.; Stehno, Z.; Vybulková, V.:* Technické prostředky a funkce mikroprocesoru 8080A. ST 12/81.
[10] *Krásný, P.; Černoch, M.:* Funkční vlastnosti programovatelného integrovaného obvodu 8255A pre paralelný vstup/výstup. ST 12/82.
[11] *Černoch, M.; Stehno, Z.; Vybulková, V.:* Funkční vlastnosti obvodu USART typu 8251. ST 5/82.
[12] *Smutný, E.:* Mikroprocesory a mikropočítače. AR B1, 2/83.
[13] *Smutný, E.:* Mikropočítačový vývojový systém JPR-1Z. AR B6/85.
[14] Mikroprocesor Z-80/programování. JZD Slušovice 1982.
[15] *Patočka, O.:* Mikroprocesor U880D. AR 2 až 8/85.
[16] *Zajíček, L.:* Bity do bytu. Mladá fronta: Praha 1988.
[17] *Černoch, M.; Stehno, Z.; Vybulková, V.:* Mikropočítač 8048. ST8/83.
[18] *Nohel, J. a kol.:* Základní instrukce/Mikroprocesor 8048. TESLA ELTOS 1986.
[19] *Zrůst, J.: Šulc, S.:* Jednočipový mikropočítač a mikroprocesor 8051. ST 1/88.
[20] *Babák, M.; Laurynová, V.:* Programovací jazyk Asembler 8051. TESLA ELTOS 1987.

(Pokračování)

INZERCE

Inzerci přijímá osobně a poštou Vydavatelství Naše vojsko, inzertní oddělení (inzerce AR B), Vladislavova 26, 113 66 Praha 1, tel. 26 06 51—9, linka 294. Uzávěrka tohoto čísla byla dne 1. 8. 1989, do kdy jsme museli obdržet úhradu za inzerát. Neopomeňte uvést prodejní cenu, jinak inzerát neuveřejníme. Text inzerátu pište čitelně, aby se předešlo chybám, vznikajícím z nečitelnosti předlohy.

PRODEJ

Mgf hlavu ANP 935 (80). J. Sauer, Za parkom 23, 990 01 Veľký Krtíš.

BFR90, 91, 96 (50, 55, 60), konekt. WK 465 80, rozteč 2,54 (80), SO42P (120). M. Pantůček, Kosmická 741, 149 00 Praha 4.

Dig. tuner Toshiba ST-G 33 (5200) a přehrávač CD TESLA MC 900 stříbrný (8300). L. Hejnal, Zápotockého 2483, 276 01 Mělník.

Tranzistory BFT66 (150) BFT97 (150), BFT96 (80), BFR90 (70), BFR96 (90). Kúpim CGY21, anténu 2038-GL starší typ aj poškodenú. P. Poremba, Clementisova 12, 040 14 Košice.

SO42 (120), BFR90 (60). P. Výborný, 569 53 Cerekvice n. L. 164.

Širokopásmový zosilňovač 40 ÷ 800 MHz, 2× BFR91, zisk 22 dB, 75/75 Ω, vhodný aj pre dialkový príjem (370), širokop. zosil. 40 ÷ 800 MHz, 1× BFR91, 1× BFR96, zisk 22 dB, 75/75 Ω, vhodný aj pre menšie domové rozvody (380). F. Ridarčík, Karpatská 1, 040 00 Košice.

V-Ω-metr 3 1/2 LCD (1000). Z. Havelka, Blažkova 8, 638 00 Brno.

7400, 04, 08, 10, 13, 22, 30, 40, 50, 53, 72, 74, 75, 81, 121, 123, 1SS1 a iné (8 ÷ 30), relé 5, 12, 24, 220 V (30 ÷ 90), mikrosp. 12, 24, 220 V (±20). Nepoužité. V zozname rôzne konektory, el. počítadlá, Xtaly, IO, T, TP, R, C, tlačítka, prepínače a iné. Len písomne. Ing. P. Štoffa, Steinerova 2, 040 11 Košice.

AM/FM budík (900), obč. rádiostanice VKP 050 (1200), A, V, Ω meter (500), zos. Zeatawatt 1420 (300), FM tuner 814A, 83, 84 (500, 700, 500), dig. stupnica 77 (500), literatúra, el. súč., AR A/B. M. Arvay, Nitrianská 13, 927 05 Šaľa.

KOUPĚ

Staré repro ARO 932 (942) ..., ø 39 cm, 15 W i bez funkce nebo bez membrány. Přijedu — po dohodě i na dobírku. P. Plevák, Svatovítská 508, 686 02 Uherské Hradiště II, tel. (632) 425 24.

Starší fungující počítač (ZX Spectrum, Didaktik, Sinclair) za 1000 Kčs (uveďte stav). R. Matuška, Fenjanská 13, 616 00 Brno.

IO AY-3-8710, dekodér PAL na TESLA Color 4401. C. Janiga, J. Kráľa 778, 015 01 Rajec.

AR B2/84, B5/85. K. Křemel, Strnadova 8, 628 00 Brno.

Geiger-Müllerovu trubici. Cenu respektuji. Ing. P. Kunce, 382 03 Křemže 118.

Hry na počítač Amstrad CPC 464. Pošlite zoznam. R. Arvay, Železničiarská 7, 082 22 Šarišské Michalány.

IO — A/D prev. do LCD Fluke 75, tov. LCD merač kapacít s automat., T, D, IO, R, C, zoznám. Záp. a sov. kat. roč. 87—89. Rôzne súč. a diely pre rôzne čb. i FTVP. J. Čižmár, Červenica 37, 082 56 Pečovská N. Ves.

RŮZNÉ

Kto za odmenu oboznámi so zapojením vývodov na IO AY-3-8765, AY-3-8605, AY-3-8603, AY-3-8606, AY-3-8607, CPQQ8010. C. Janiga, J. Kráľa 778, 015 01 Rájec.